ORTA DOĞU TEKNİK ÜNİVERSİTESİ
MIDDLE EAST TECHNICAL UNIVERSITY
KUZEY KIBRIS KAMPUSU ◆ NORTHERN CYPRUS CAMPUS

CNG 492 Report | Final Report

Project Name:

Optify|Optimization System for Scheduling Routine

Team Members:

Orkhan Salahov 2193191
Rustam Quliyev 2250470
Rasul Aliyev 2250462

Table of Contents

1. Introduction & Literature Review

How many times have you caught yourself trying to find a timeslot for a meeting within a small group, which would suit you all? If you are not an alien to such problem then most probably you have experienced lots of back and forth chatting to find out the timeslot and the place of your meeting. We aim to automate this process with the help of cutting-edge AI algorithms. Optify is an optimization system for scheduling routine that would find the most optimal timeslot/location of the meetings according to attendees' needs. We imagine our platform as a social network that would facilitate people to help with their routine scheduling.

During the course of our graduation project we are aiming to finish the Scheduling part of the project for single and multi-attended activities which will aid the users with the datetime of activity. Also we planning on finishing Mobile application for iOS & Android platform with working back-end connection.

1.1. Background Information

There are many tools in the market that provide a very neat way to organize ones tasks throughout the day. Such tools provide services such as creating a schedule of activities, sharing activities amongst your friends, tagging different activities and etc. However, even with such an abundance of scheduling tools, people still do spent lots of their time & effort on organizing and picking appropriate timeslot for their activity. Especially the problem revels its full complexity as the number of attendees grow. In such scenario there are many things to be considered while picking a suitable timeslot. Each users' current calendar, his/her preferences, the weight that person has in the activity, his/her working & sleeping cycle and etc. To make things worse the attendees can be in different time zones.

There are a few researches that have been done on this matter. Mitchell et al. proposed Calendar Apprentice (CAP) which is a decision tree based calendar manager that can learn user scheduling preferences from experience [1]. In this research their main focus was trying to learn user preferences in schedule by using ANN and Decision Trees. They came to conclusion that Decision Trees is preferable for 2 reasons. Firstly, it is more human tracible, since the conditions are quite intelligible by humans to approve or reject. Secondly, because of the nature of decision tree the conditions are piecewise functions, which means that system can evaluate each rule individually. Then Blum et.al has introduced Weighted Majority and Winnow algorithms which outperformed the previous Decision Tree Method[2]. In this paper she displayed the use of latter algorithms in the scheduling of seminars with lots of people attending. A different approach was developed by Mynatt et al.[3]. He proposed to use the schedule of a person to determine his whereabouts at a given time. Of course not all activities in our schedule get attended that is why he used Bayesian models to predict the attendance of an event in the schedule. Finally as my literature research goes Donghyeon Kim et al. have

critiqued most of the previous approaches saying that they are very obsolete in its restrictions such as: event types, attributes and heuristics.[4] Instead they have developed fully Deep Learning based algorithm that would correctly guess an entry in users calendar by training with Public Google Calendar data. There they offered to use NLP to extract the intention of an activity through its title, which in their opinion would also account for the output.

Having studied the literature about the subject so far it is determined that the task of predicting an activity starting time should account for the users' preferences as well as the intention of the activity. For example if a group wants to arrange a joint activity with title 'Dinner with friends' the algorithm should consider only evening times since it is a dinner. However even training Deep models can be quite a mess since the schedule data publicly available are not a good example of optimal timeslots. In other words not all the data that is in ones' schedule is his/her preferred timeslot. Therefore even collecting appropriate data for the model is a nontrivial task.

2. Identification of Constraints

There are no serious constraints available to the system, however couple of them are mentioned below.

Psychological

Not all people are feeling comfortable filling in their schedules and planning their day through an app. We believe that number one reason is the data rate problem. Adding an activity to schedule seems to be a very monotonic process. To overcome this we may add new features to the app such as integration with Voice assistants like Siri, Alexa and etc. However, it was not within the scope of the project.

Security

In the application people can make different privacy settings for the activities, it is vital for the reputation of the app to not have any holes where one user can view private activities of another user.

3. Requirements

2.1. System requirements specification

2.1.1 Functional Requirements

User Requirements

1. The application should have a usable, intuitive Graphical User Interface (GUI) with vivid, inspirational colors similar to that of Material Design.

Name	Intuitive GUI
Description	Most people have lots of stuff happening within a day, hence their schedule gets quite messy. We think that it is vital to have user-friendly UI for such app
Inputs	*not applicable*
Outputs	*not applicable*
Importance	High

2. Application should prompt the users to sign in or register before starting to use the application.

Name	Log in
Description	Users will need to sign-in to use that app
Inputs	{Email, Password}
Outputs	System Activity point
Importance	High

Name	Sign Up
Description	Users will be able to register to the system
Inputs	{Email, Password} AND Optional{Name, Surname, Profile pic, status}
Outputs	Login Page
Importance	High

3. During the first use of application the user should be, optionally, prompted to update his/her schedule to the app. It can be edited later anytime through the app's corresponding section

.

Name	Add Activity to Schedule
Description	System should have full scheduling functionality through which users can add Activities.
Inputs	Title, DateTimes, type, priority, Timing and optional arguments
Outputs	Updated Schedule
Importance	High

Name	Scheduler AI
Description	User can use the scheduler AI to determine the best timeslot for the activity.
Inputs	Constraints, Button click
Outputs	Top Timeslots
Importance	High

Name	Import Schedule
Description	Since nowadays most people use Google Calendar and Apple Calendar users can simply import them instead of rewriting
Inputs	Google Calendar OR Apple Calendar credentials
Outputs	Updated Schedule
Importance	High

4. The user can edit the apps for which the notifications will be blocked during the time of Do not Disturb regime. By default all apps should be blocked.

Name	Do not Disturb
Description	Activities will have a do not disturb checkbox, hence there should be a place to control which apps to block during that time.
Inputs	App List
Outputs	Updated Settings
Importance	Low

5. Activity types should have defined tag colors.

Name	Customize types
Description	Since Activities will have some types and those types will have colored tags. User should be able to customize those types and colors as they wish.
Inputs	Type Color Map
Outputs	Updated Types
Importance	Low

6. User can view his/her calendar in a separate section of the app.

Name	Normal Viewing Mode
Description	There should be normal viewing mode for the calendar where users can view their Activities colored as their type color.
Inputs	Toggler
Outputs	Normal Calendar View
Importance	High

Name	Priority Viewing Mode
Description	There should a mode for viewing the Activities as heatmap according to their priorities.
Inputs	Toggler
Outputs	Priority Calendar View
Importance	High

7. User can view, update, delete Activities in the calendar.

Name	CRUD Activities
Description	User can CRUD his Activities in the calendar.
Inputs	Corresponding Button
Outputs	Updated Activity
Importance	High

8. There will a section in the app to display statistics about ones routine.

Name	Statistics
Description	A user can view his/her statistics in the corresponding page.
Inputs	Tab Switch
Outputs	Statistics View
Importance	High

9. Users can add/view/delete contacts.

Name	CRD Contacts
Description	Since our app should be social users are able to send contact request, view and remove contacts.
Inputs	Contact Name Surname
Outputs	Contact
Importance	High

10. Users can CRUD their profile and view others' profiles

Name	Profile Page
Description	Users should be able to view their or others' profile page. For their own ones they can also remove or update them.
Inputs	Button click
Outputs	Profile Page
Importance	High

11. Privileged members can share the activity

Name	Share Activity
Description	Owner and the privileged users can share the Activity in-app or using the Activity link.
Inputs	Button
Outputs	Share
Importance	High

12. User can CRUD Groups

Name	CRUD Groups
Description	Users can form groups with their contacts.
Inputs	Button
Outputs	Share
Importance	Medium

13. Groups have own schedules, where privileged ones can add a new activity.

Name	Group Schedule
Description	Privileged ones can add Activities to group schedules.
Inputs	Activity
Outputs	Update Group Schedule
Importance	High

14. Activity will have participant and tracking list.

Name	Activity Participant List
Description	In Activity's participant list everyone can see all other participants and their roles.
Inputs	Button
Outputs	Participant List
Importance	High

15. There will be chat section in the app.

Name	Chat
Description	There shall be a button in the group only available to admin which will create a chat within the group members. Also user to user chats will also be available.
Inputs	Button
Outputs	Created Chat
Importance	High

16. There will be Settings section within the app.

Name	Settings
Description	App will have general settings as well as specific to group, profile, Activity and activity.
Inputs	*not applicable*
Outputs	*not applicable*
Importance	Low

17. There will be a list of activities.

Name	Activity List
Description	User can list his/her activities using different filtering methods.
Inputs	Button
Outputs	List
Importance	High

System Requirements

1.1. The GUI should be designed similar to the mock-up.

2.1. User must provide password and email in order to register.

2.2. An email should be assigned to only one account.

2.3. During registration the user can provide his/her Name, surname, profile photo, status.

2.4. An email with the validation link should be sent in order to validate the account.

2.5. Upon login the user can enter either email or username with his password.

*2.6. The app should support registration through Facebook and Google Plus(*requires app registration).*

2.7. While registering through Facebook or Google Plus the app should provide the list of ones friends that are using the app and a button to add them to your friends in the app. This step can be skipped if wished so.

3.1. An activity for the schedule shall have a title. The name of the Activities are not unique.

3.2. An activity for the schedule shall have one or many starting and ending datetimes, in other words for each activity multiple blocks with starting and ending datetimes may exist.

3.3. Based on previous activity records the ending time should be calculated after setting the starting time of a block of time in the activity to aid the user.

3.4. An activity for the schedule shall have a type. The predefined types are: Default, Academic, Sport, Recreational, Personal, Social and Work.

3.5. An activity for the schedule shall have a certain priority number which is a float between 0 and 100.

3.8. An activity for the schedule shall have a privacy setting, which are one of the following: Public, Friends, Private, Exceptions… Default being Friends. By setting custom privacy user can select any combination of groups and friends to whom the activity will be visible.

3.9. An activity for the schedule shall have notification settings. The notification settings should select the number of minutes/hours/days before which to notify the user about the Activity. Default notification should be with in-app notification 30 minutes before the activity.

3.10. The user can select means of notification which can be in-app notification, mail or SMS.

3.11. The location can be set for the schedule as well. There should be a button which will display the map to select the location by hand or by typing in the details of the address to the requested field.

3.12. An activity for the schedule shall have a do not disturb checkbox which won't allow any external notifications to pop-up during the time of the activity.

*3.13. Schedule can be imported from conventional Calendar apps, namely: Google Calendar and Apple Calendar(*Requires app registration).*

3.14. Schedule should support export & import in ICalendar specification

3.14. Activities can optionally have a description as well.

3.15. Activities can be added either by clicking on an empty space in the calendar or by clicking the add button.

3.16. Activities can be easily dragged around in the calendar which will cause a shift in its time. The shift shall be in increments of 15 minutes.

3.17. Activities datetime can also be determined using the Scheduler AI.

3.18. Activity can be fixed time or live time. Fixed time Activities are fixed once and cannot be changed whereas live time Activities will be changing each time new activity is added with the help of Scheduler AI.

3.19. User can fix the timeslot of any live activity anytime.

4.1. General settings can be reached through the settings section of the app.

4.2. However, for each activity user can, optionally, add or remove certain apps from this list. Database should store activity specific apps list in an efficient way.

5.1. Each activity type shall have an assigned color to it.

5.2. Predefined type colors are: Default: black, Academic: blue, Sport: green, Recreational: yellow, Personal: white, Social: red, Work: purple and Activity: Turquoise.

5.3. User can add or remove types as well as reassign different colors to the types in the settings menu.

6.1. There will be a section in the app to view the calendar.

6.2. Calendar can be viewed on, daily, weekly, monthly, yearly basis as well as a schedule.

6.3. Calendar shall support 2 viewing modes: normal and priority. In normal mode Activities will be rendered in the color of the type. In priority mode Activities will be displayed in a shade or red based on their priority level. Hence, the higher its priority the more reddish will the activity appear.

7.1. User can view the details of any activity, as a pop-up, by clicking on it in his calendar.

7.2. From the pop-up user can delete or edit the activity.

7.3. All the parameters are editable by the user.

8.1. There should be a general statistical variable showing the global attendance rate in main window.

8.2. There should be a pie chart displaying the time one spends on each type of activity.

8.3. User can search for a specific activity and get information about attendance rate and spent time.

9.1. Contacts can be searched using their Name and Surname.

9.2. One can send friend request for those who are not your friends yet. If both parties agree to connect they will be updated as contacts.

9.3. Users can view their current contacts, and search through them.

9.4. Users can remove their contacts anytime.

10.1. Users can view their own profiles. As well as they can Update information and remove their own profiles.

10.2. Users can update their name, surname, description and avatar.

10.3. Users can view others profiles. There they can see their map, schedule and combined schedule.

10.4. Users can have status as Public, Only Friends, Inactive.

11.1. Privileged users can invite contacts to the activity in-app.

11.2. If the settings allow, privileged users can copy the activity invitation URL and send to anyone interested.

11.3. A person following the link will be in the activity invitation page where he/she can view all the activity details with the Accept button.

11.4. If the user has accepted the invitation once he shouldn't be viewing the page again.

11.5. Accepted activities will be put in the schedule of every user.

11.6. Admin has all the privileges.

11.7. Any invited user can leave the activity anytime.

11.8. Owner of an activity can chose whether guests can add more guests as well.

11.9. URL can be on-time-link or public link.

12.1. Groups have name, profile pic and description.

12.2. Privileged ones can add/kick members to the group.

12.3. Groups can have their schedule as well as joint schedules of all the users.

13.1. Groups provided members can add new Activities to group calendar.

13.2. Group activities will be just regular activities with group members attending it.

14.1. Participant List can be viewed by anyone participating. Their roles as well as their names shall be visible.

14.2. Participants can be individuals as well as groups.

14.2. Privileged participants can alter the roles of other participants, as well as assign priorities to each of them.

14.3. For the public activities there will be a request to join list which is moderated by the privileged users. Those participants can accept or kick users from participation.

15.1. Only admin of the group can create a chat. It will be created optionally once for each group.

15.2. Users can send message to any of his/her contacts.

15.3. Chat should support simple emojis.

16.1. In Settings Section User can set default Calendar Privacy Settings

16.2 In settings sections user can delete his profile

16.3 In settings sections user can set default privacy parameters for any activity

17.1 Users can see Activities for a week as a List

2.1.2. Non-Functional Requirements

Usability	• System should have a very intuitive design with many hints along the way. • While using the Scheduler AI the system should provide detailed feedback about the quality of the constraints.
Reliability	• The system should be error tolerant, by detecting fallacious or malicious data • Scheduler AI should be deterministic and try to minimize randomness in the algorithm.
Scalability	• System should be developed in highly modular way to add-up new features as we go. • System should be designed to handle large amounts of data.
Availability	• System should have 5 nines availability.
Performance	• System should be flawless in basic Input/output operations • Response time on average should be less than 3 seconds. Since less than a second might not be feasible but the users shouldn't lose their focus.

Internalization	• The system should support flawless transitions between different timezones

2.1.3. Domain Requirements

1. System should consider different time zones while deciding for the available slot and displaying the calendar.

2.2. System Models

2.2.1. Stake Holders

Rasul Aliyev – 4th year Computer Engineering student at METU NCC.

Rustam Quliyev – 4th year Computer Engineering student at METU NCC.

Orkhan Salahov – 4th year Computer Engineering student at METU NCC.

2.2.2. Structured Use Cases

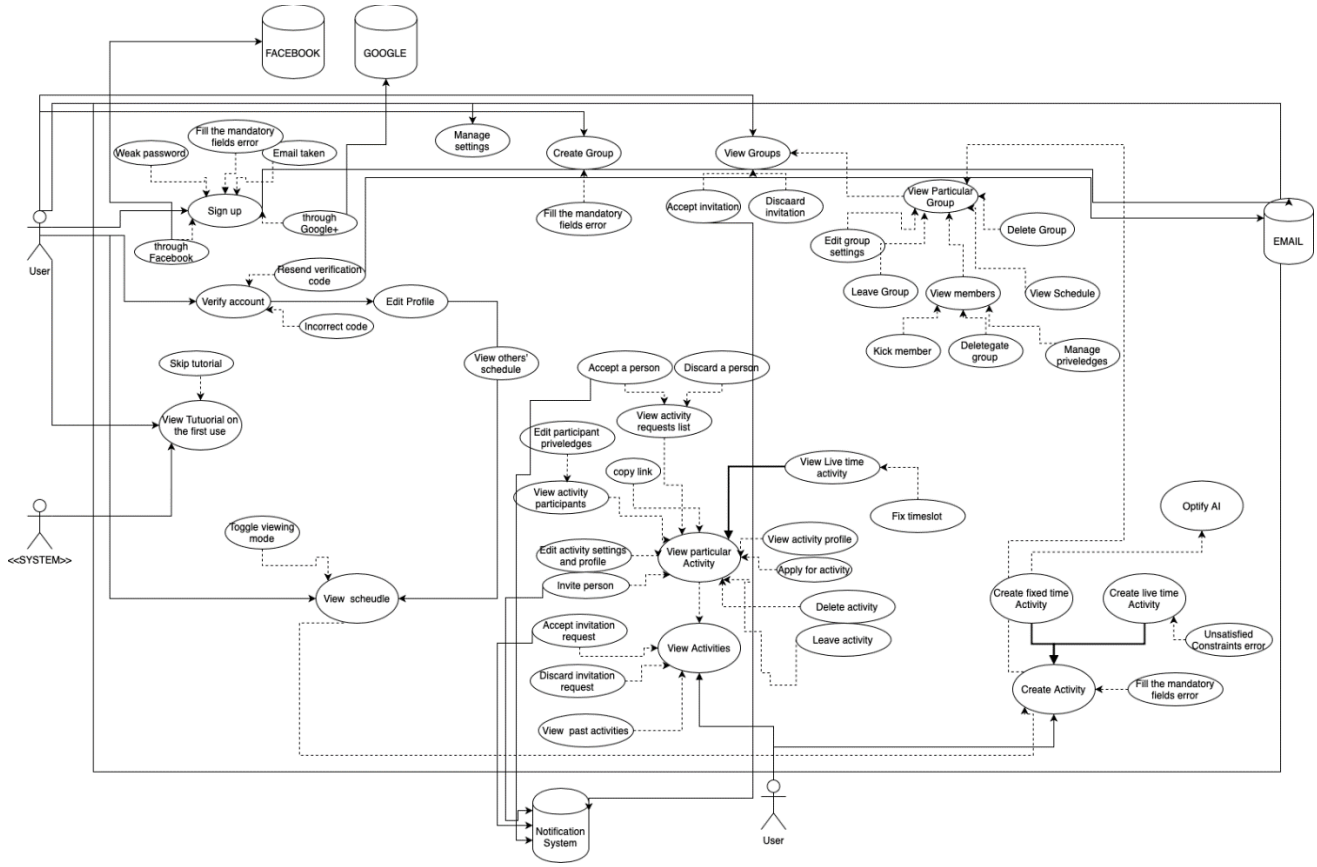


Figure 1 | Use Case diagram Part 1

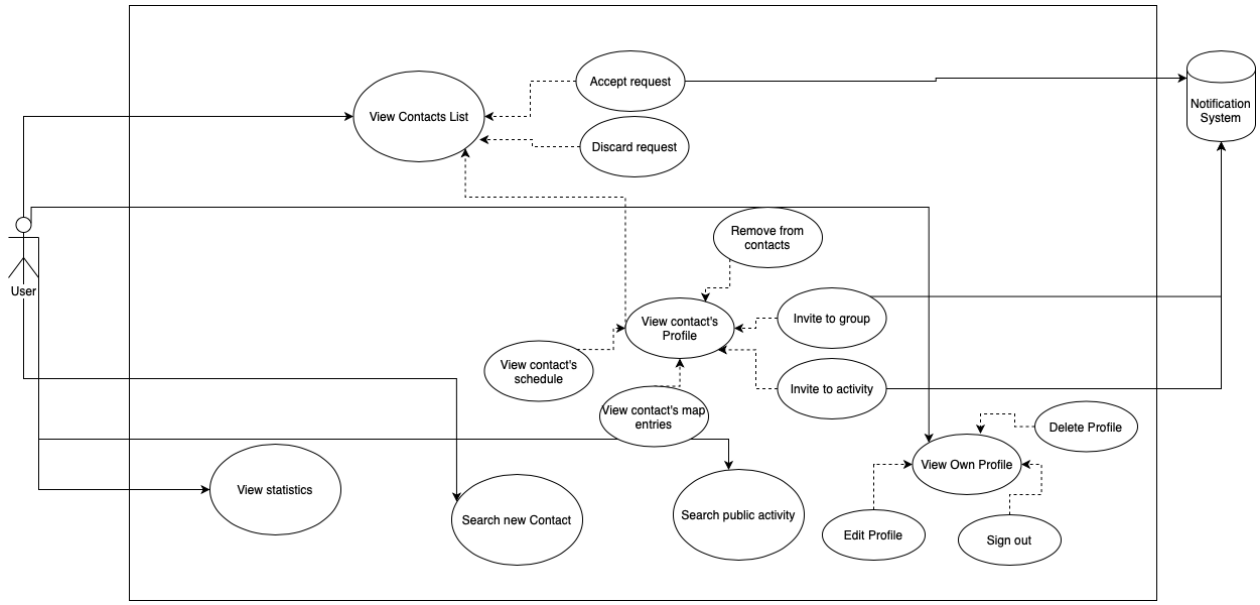


Figure 2 | Use case Diagram Part 2

https://drive.google.com/file/d/16eOKG_XfOik4K-JiralqPEN9CKlZ5w4j/view?usp=sharing

2.2.3. High Level Sequence Diagram

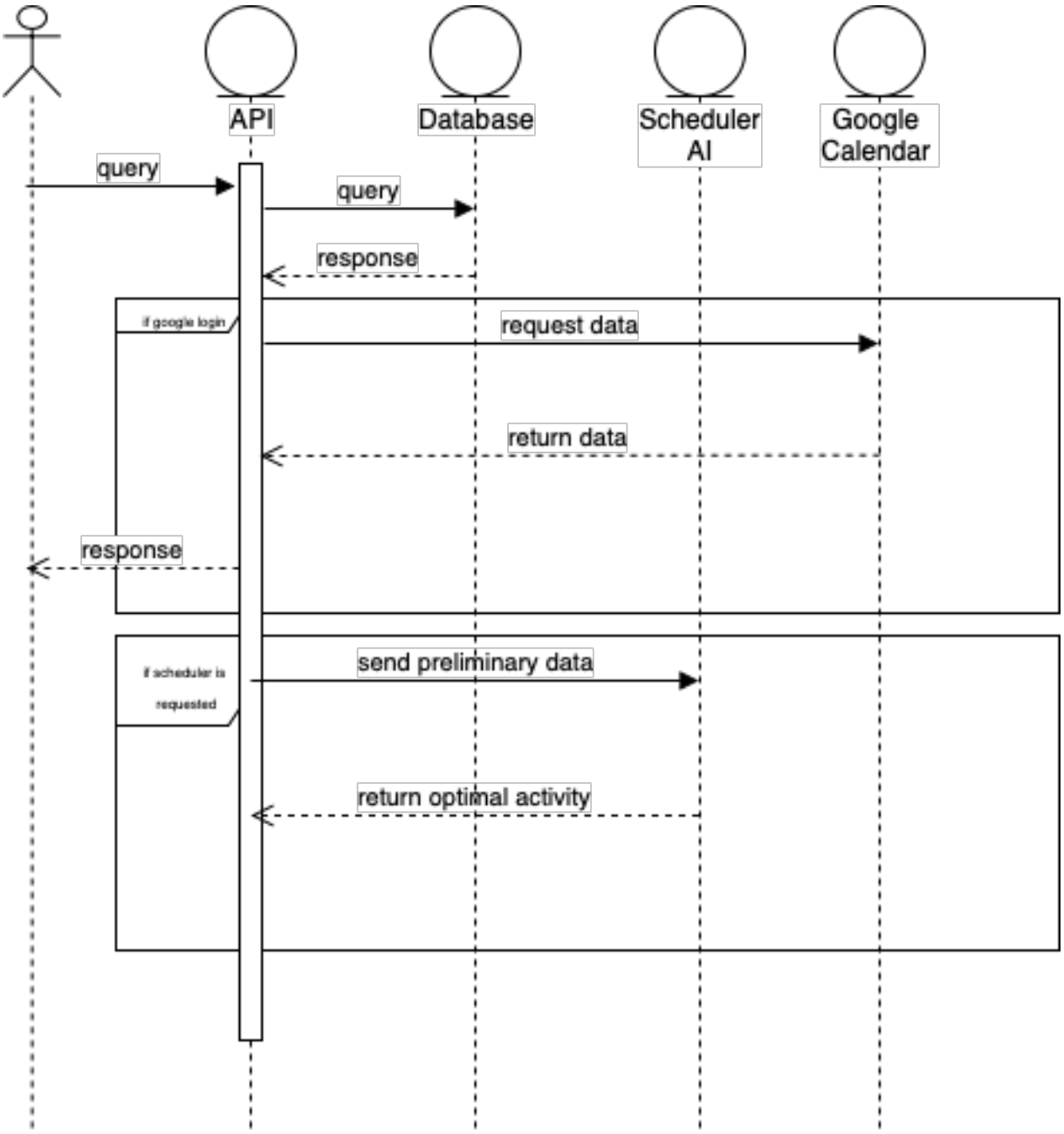


Figure 3 | High Level Sequence Diagram

2.2.4. Context Models & High Level Architecture Diagram

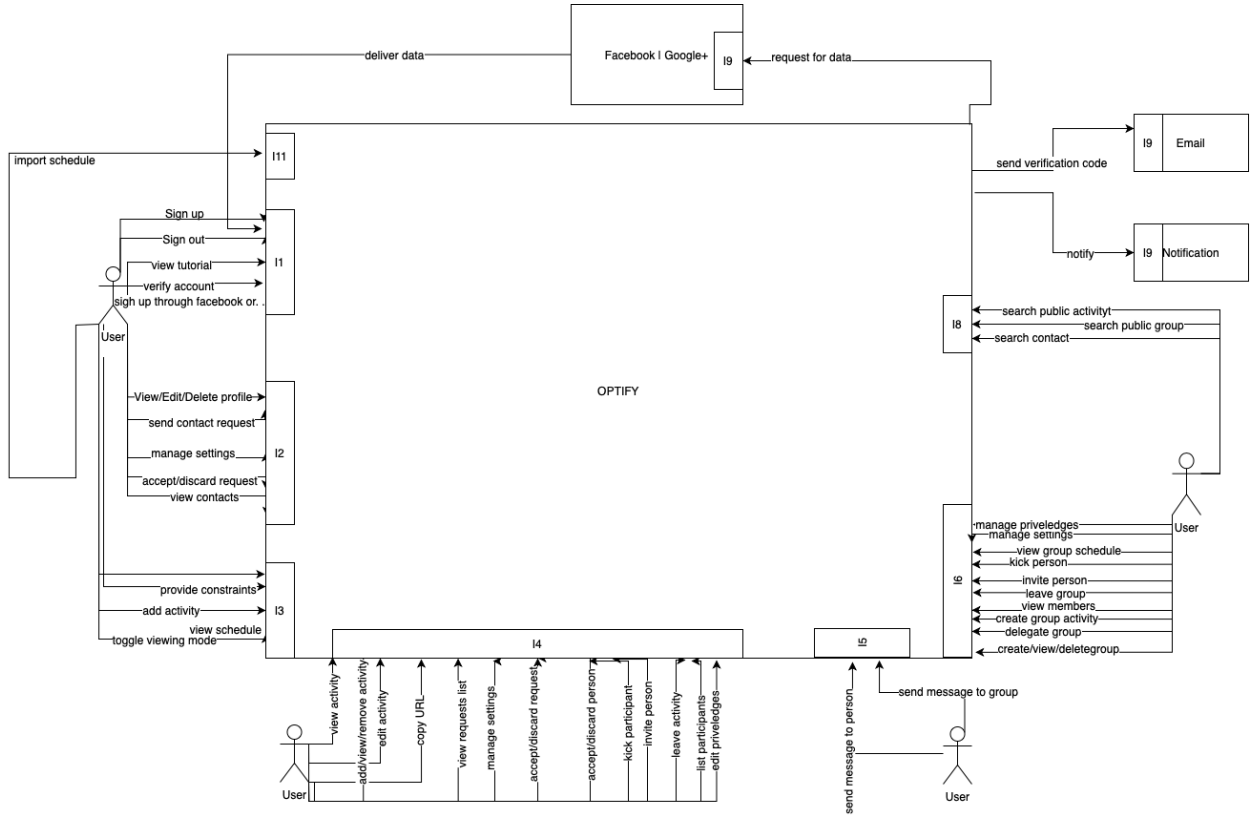


Figure 4 | Context Diagram

2.2.6. Process Models

Data Flow Diagram for Creating an Activity

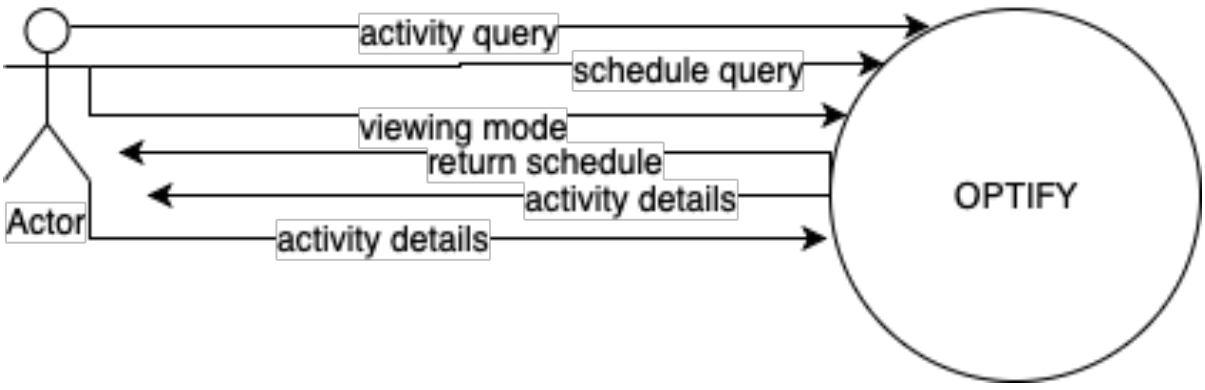


Figure 5 | DFD Level-0

2.2.7. Interface Definitions

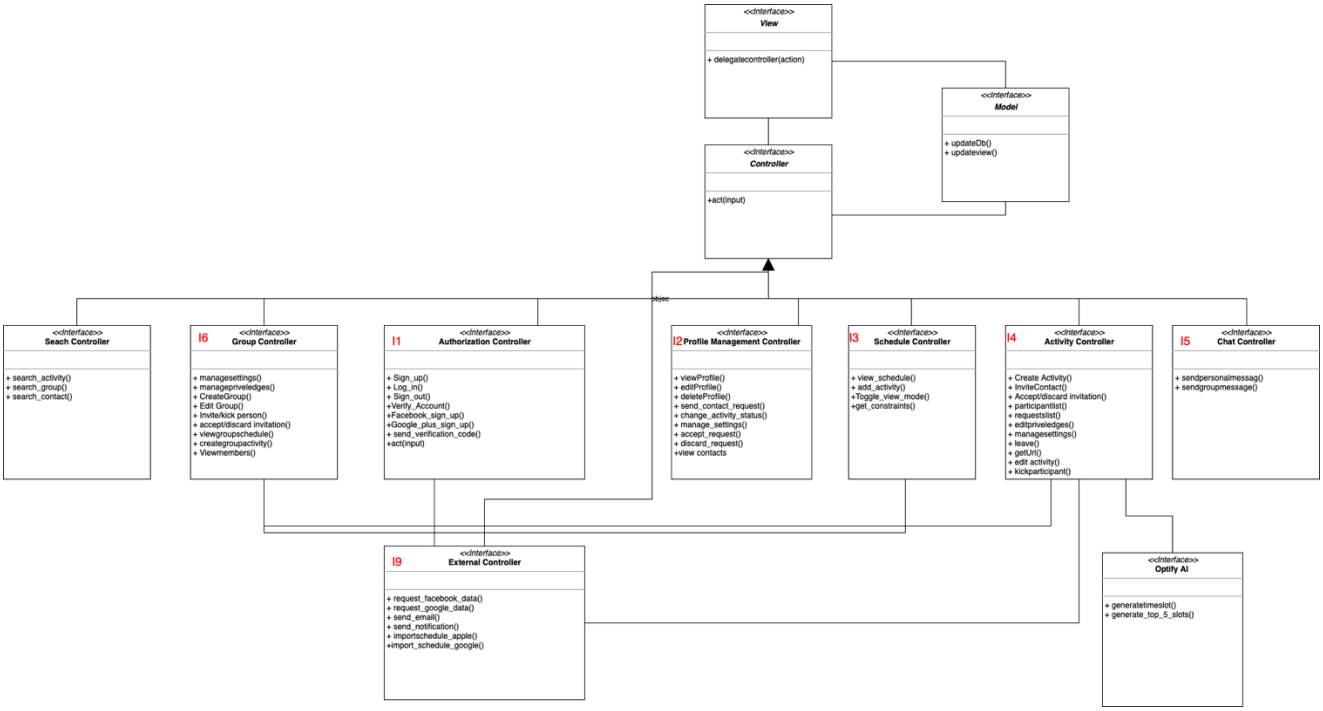


Figure 6 | Object Model

https://drive.google.com/file/d/1EyMfF6xzTwfV1Gj3srR0Cr5qrqq6rLz9/view?usp=sharing

2.3. Test Requirements & Test Plan

Test Plan

Flutter Application

Participants: Rasul, Orkhan

For the Flutter application is composed of bunch of widgets(Views) with intertwined Back end API calls. Flutter offers **3** testing facilities for this purpose, which are: **Unit Test, Widget Test, Integrations test library.** We are going to make all the 3 types of test. Unit tests will be applied to main functions which mainly deal with the API or have core role in the Widget. The goal of a widget test is to verify that the widget's UI looks and interacts as expected, and Integration tests will help to validate cross-dependency errors and etc.

Back-end Software

Participants: Rustam

For the REST API testing we going to use the **Soap UI** framework. This framework is going to manage all (3+1) types of testing that we are going to have in our back-end. Those are **Unit**

testing the individual views, **Load Testing**, and **Security Testing**. Sometimes if we would need fake data in large quantities we are going to use ready tools such as **Faker**.

Requirements

Unit Testing

Brief	Item	Participant	Methodology
Create User API Call	Create user API	Rustam	Fake user data will be populated and new Users will be created thus seeing if all the field are properly validated. (**Faker + Script**)
Delete User API Call	Delete User API	Orkhan, Rasul	Users will be deleted through the app after having some data in the DB to check if no dependency error appear. (**Soap UI + Manually**)
Create new Activity API Call	Create new Activity API	Rustam	Fake Activity data will be generated to populate Activities for one user. Thus checking how the app responds to invalid fields and etc. (**Soap UI**)
List Activities in Application	Calendar View	Orkhan, Rasul	Users Activity data should be fetched and correctly displayed in a Calendar View(Recurring activities, associated tag colors timezone) (**Manual**)
Update Activity in Application	Update Activity Info	Orkhan,Rasul	User will Update its activity information using app thus seeing if output is as expected for all the guests. (**Manual**)
Delete Recurring Activity	Delete Recurring Activity	Orkhan,Rasul ,Rustam	Delete any Recurring activity in the app. Check if app gives correct

			prompt message. Delete the 'whole' activity or 'instance' only. (**Soap UI + Manual**)
Invite Guest to an Activity in Application	Invite Guest	Rustam	User will add multiple guests to the activity in the application. Check if the correct prompt message is sent. (**Soap UI**)
Check Live activity changes after Invite is accepted	Live activity time	Rustam, Orkhan, Rasul	After user has joined any live time activity the activity time should be changed in accordance to the joined user. Test that this is happening. (**Soap UI + Manual**)
Google Calendar Import	Google Calendar Import	Rustam	Import from Google Calendar data for a User. Check so that correct DB fields are populated. (**Manual**)
Create Group through App	Create Group	Rustam	Create a group through app by having several members. Check so that all invalid data are discarded and upon creating a Schedule for group is created. (**Soap UI + Manual**)
Kick/Add new Members from group	Kick/add new	Rustam	Kick a User from the group, by validating none of his trace is left in the group activities. Upon adding check so that all live group events are rearranged. (**Soap UI + Manual**)
Escalate Participants Privileges in Group	Participant Priveledges	Rustam	For all of the possible privileges assign each of them one by one verifying the correct restrictions are given in the API. (**Soap UI + Script**)

Create Group Chat	Group Chat	Rustam, Orkhan,Rasul	Verify that only admin can create Chat for the group. Verify that chat gets populated with correct members. And is straightly accessible. (**Soap UI + Manual**)
Message Socket Connection	Message Socket	Rustam, Rasul, Orkhan	Upon entering to a Chat Room application should connect to API via **WS** Protocol. Verify that this happens smoothly. (**Manual Debugging**)

Widget Testing

Brief	**Item**	**Participant**
Draws the boxes in the Weekl Calendar screen	BoxObject widget	Orkhan, Rasul
Draws the activity item in the list of the activities page	ActivityListElement widget	Orkhan, Rasul
Draws the activities inside the Weekly Calendar screen	Draw Activity widget	Orkhan, Rasul
Side drawer on every page with navigations.	Drawer widget	Orkhan, Rasul

Draws the bottom navigation bar in all Pages	BottomNavBarwidget	Orkhan, Rasul
Draws the individual chat users in Chat Screen Page	ChatItem widget	Orkhan, Rasul
Draws the individual message items in Chat Window Page	MessageItem widget	Orkhan, Rasul
Shows user details in Profile Screen	Profile Screen widget	Orkhan, Rasul
Shows statistics about the user in Statistics Screen	StatsScreen widget	Orkhan, Rasul
Displays connected members and activities in Groups Screen	GroupsScreen widget	Orkhan, Rasul
Displays friends of user in Contacts Screen	ContactsScreen widget	Orkhan, Rasul

Displays upcoming activities in Activity Screen	UpcomingActivities widget	Orkhan, Rasul

Integration Testing

Participants: Rustam, Rasul, Orkhan

After the mobile application has been built & tested for its units we are going to combine all pages in one and test the integration, the whole app.

Stress Testing

For the stress testing first we are going to the **Baseline Testing** which is the performance of the system under normal load(say 1 request per second). Then we are going to do the actual **Stress Testing** and find the system break point using the **Soap UI** tool.

Acceptance & Performance Testing

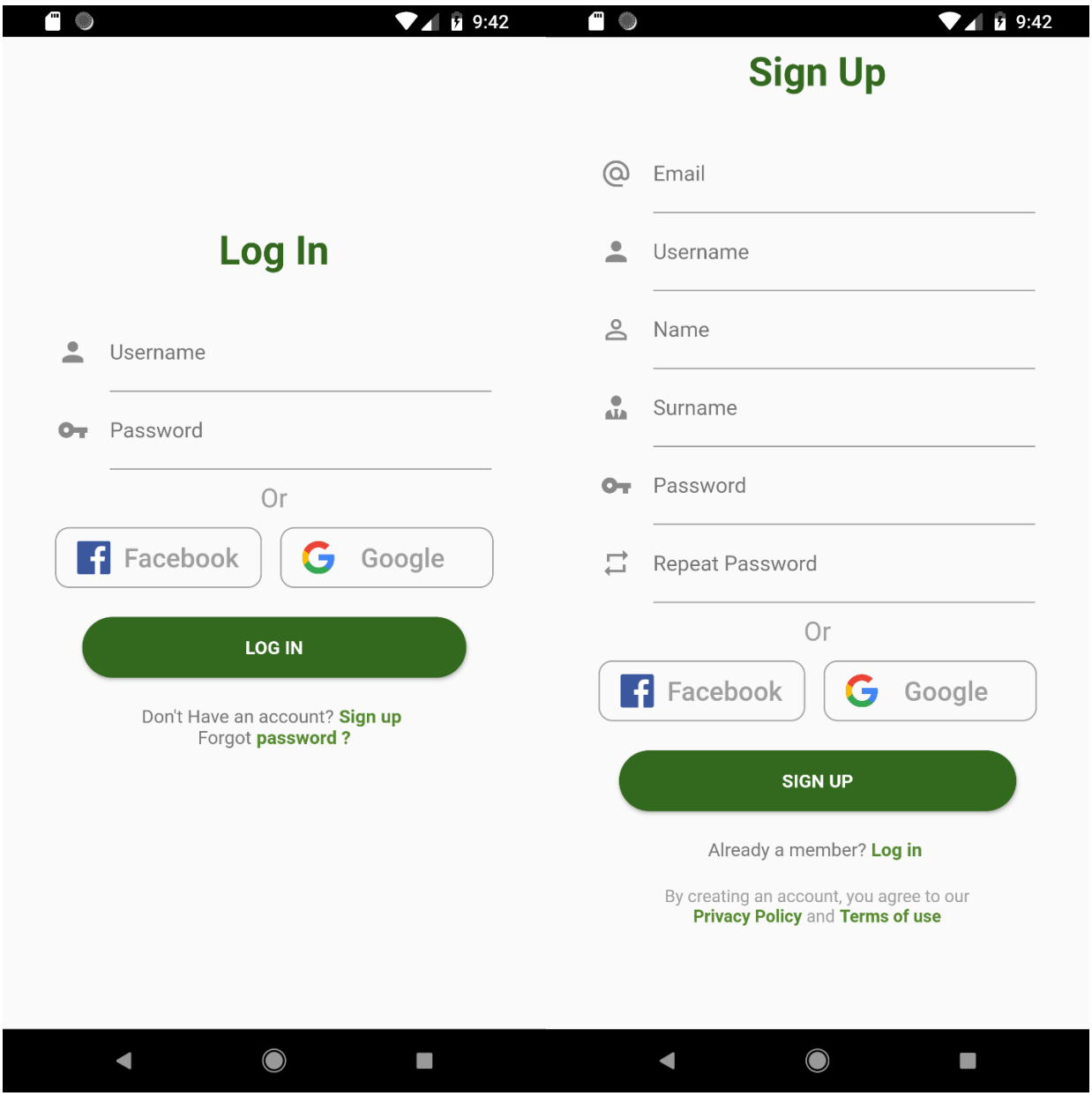
- Make sure that the Calendar View is going on seamlessly, Virtual List to be Used for easy transitioning.
- Application loading time < **5s**
- Network Latency for primitives(CRUD) < **500ms**

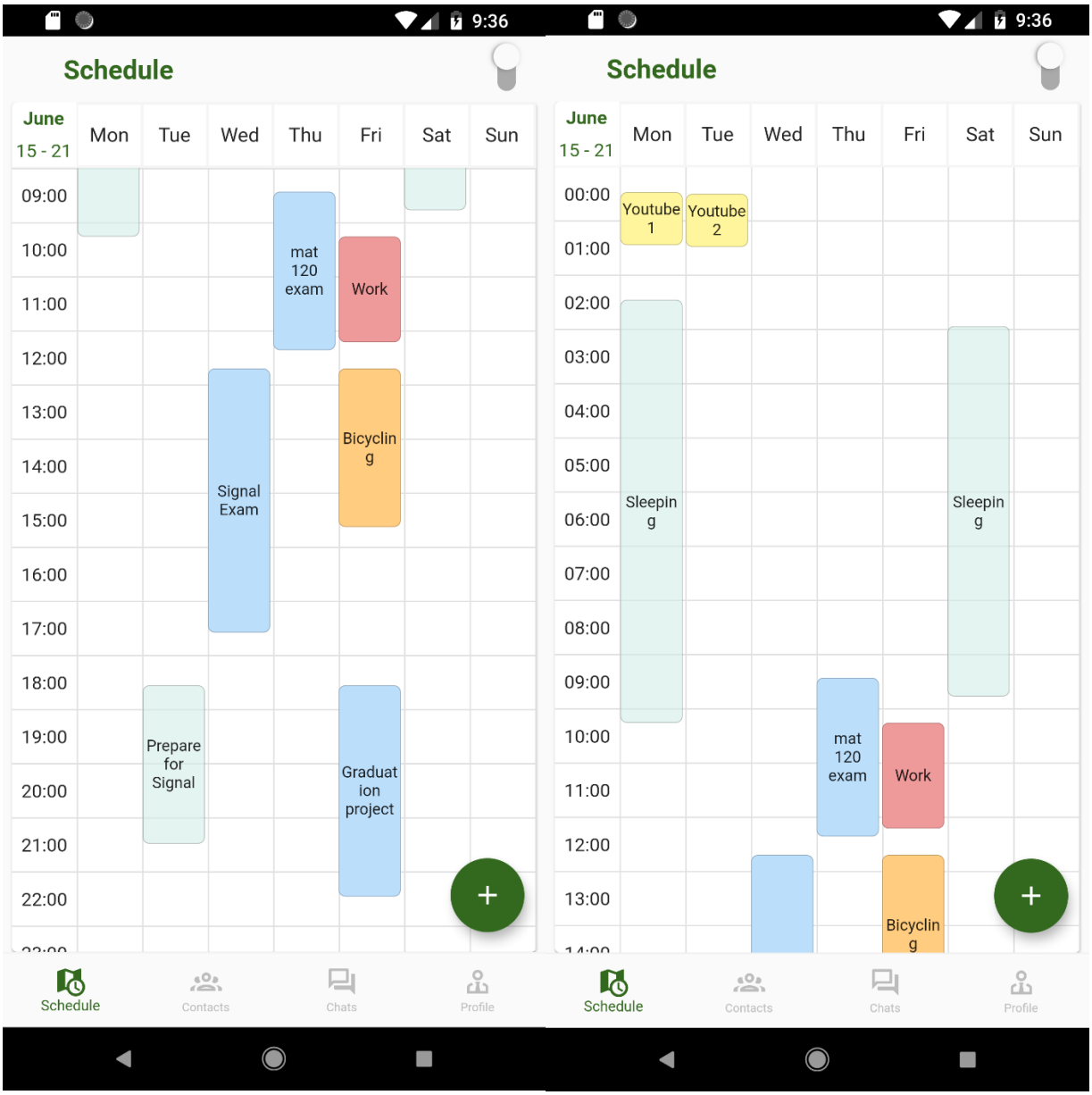
Security Testing

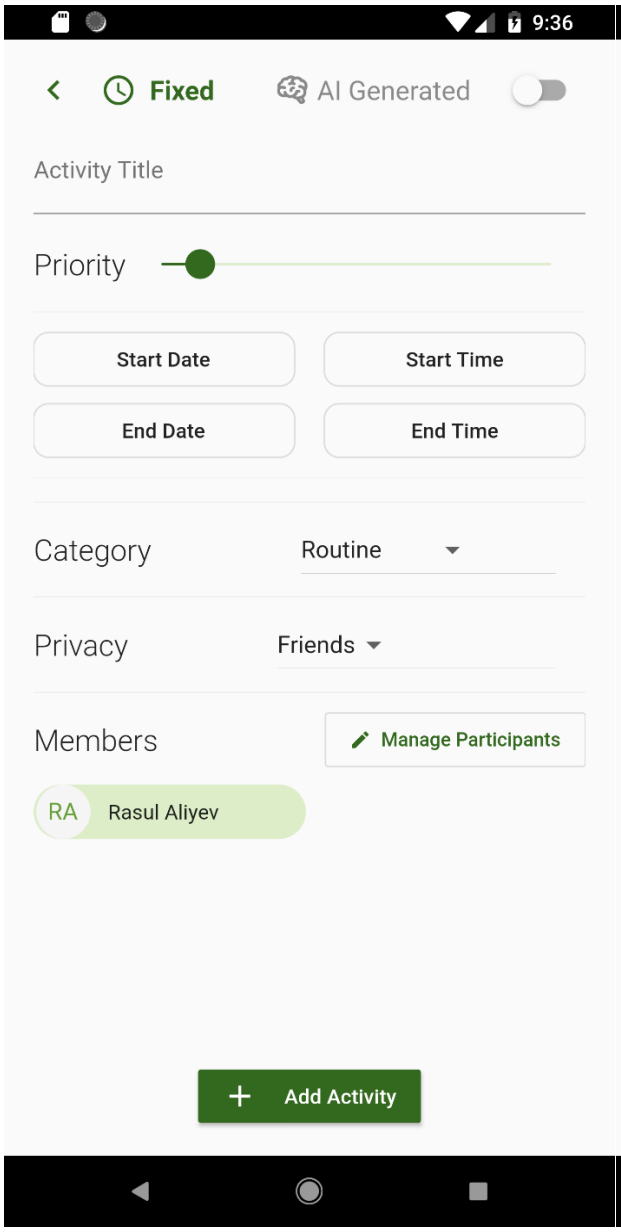
For the back end security test as we have mentioned SoapUI is going to be used. We are going to generate test requirements for testing the common security vulnerabilities below. All of these are done using **Soap UI** tool.

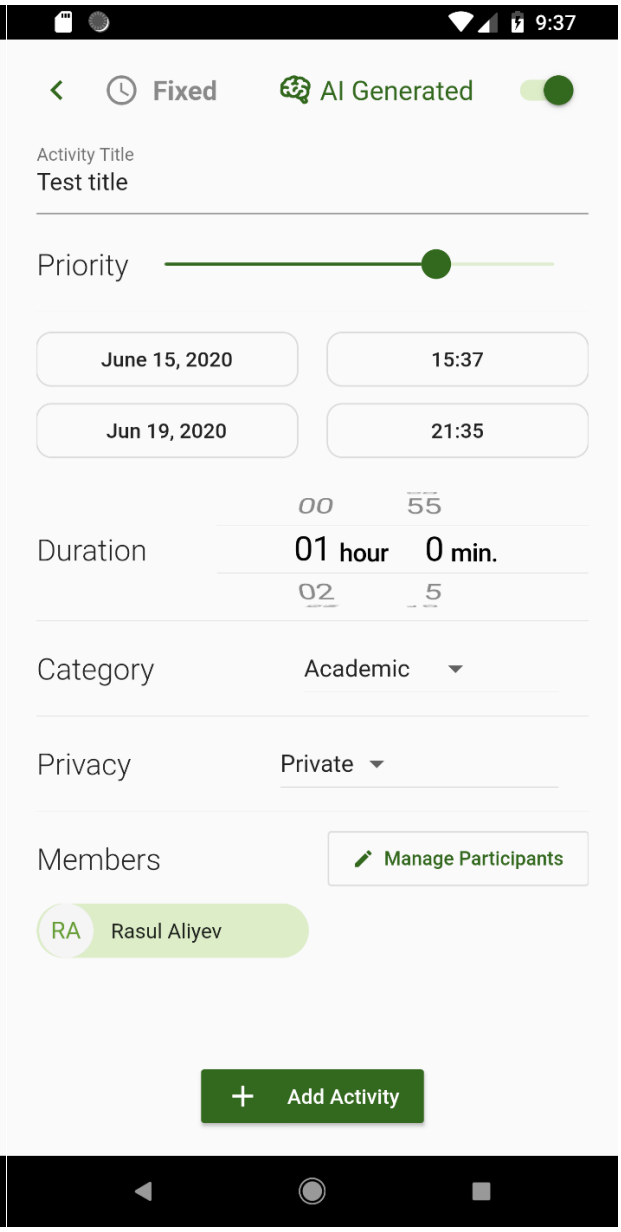
- SQL Injection : tries to exploit bad database integration coding
- XPath Injection : tries to exploit bad XML processing inside your target service
- Boundary Scan : tries to exploit bad handling of values that are outside of defined ranges
- Invalid Types : tries to exploit handling of invalid input data
- Malformed XML : tries to exploit bad handling of invalid XML on your server or in your service
- XML Bomb : tries to exploit bad handling of malicious XML request (be careful)
- Malicious Attachment : tries to exploit bad handling of attached files
- Cross Site Scripting : tries to find cross-site scripting vulnerabilities

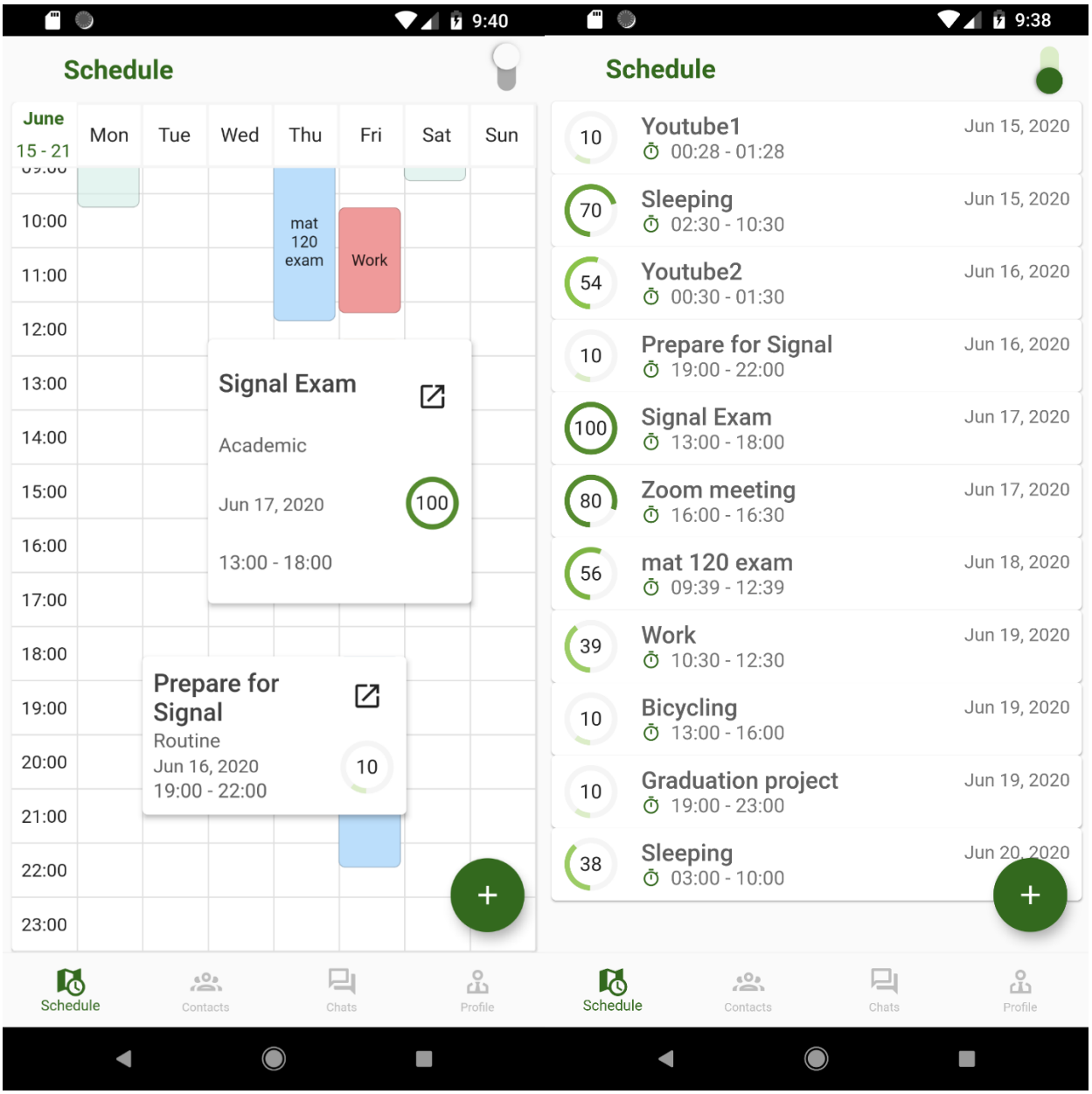
2.4. Graphical User Interface

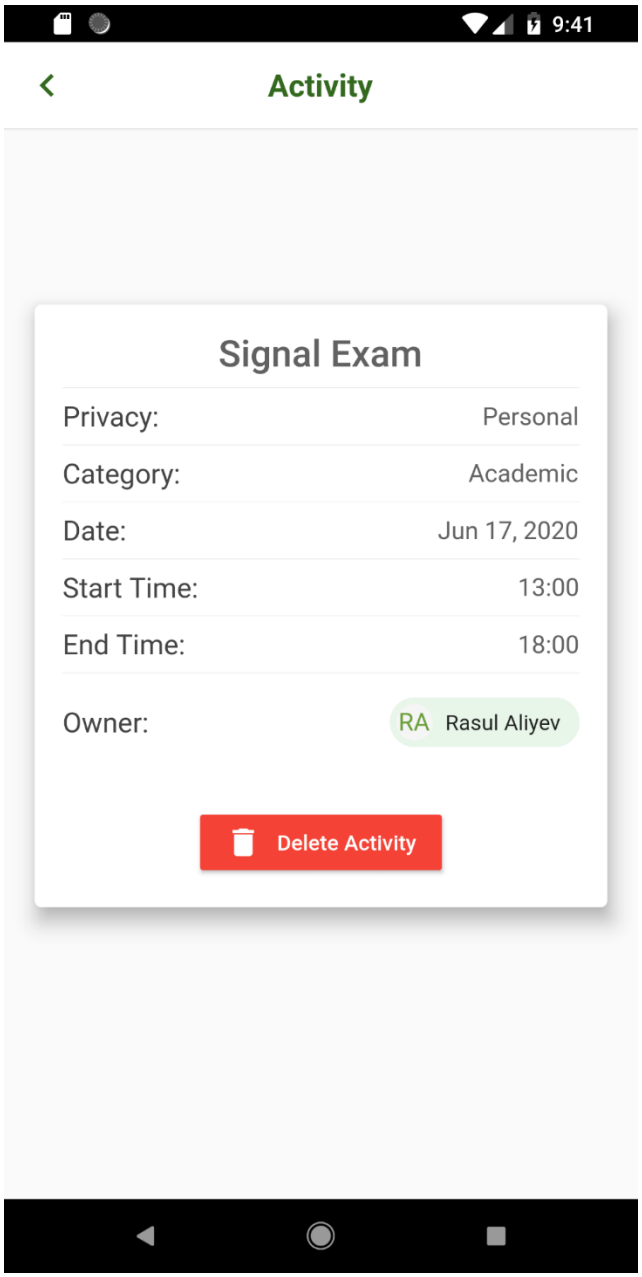



9:36
Schedule
June
15 - 21
Mon
Tue
Wed
Thu
Fri
Sat
Sun
09:00
10:00
11:00
12:00
13:00
14:00
15:00
16:00
17:00
18:00
19:00
20:00
21:00
22:00
mat 120 exam
Work
Signal Exam
Bicycling
Prepare for Signal
Graduation project
Schedule
Contacts
Chats
Profile
9:36
Schedule
June
15 - 21
Mon
Tue
Wed
Thu
Fri
Sat
Sun
00:00
01:00
02:00
03:00
04:00
05:00
06:00
07:00
08:00
09:00
10:00
11:00
12:00
13:00
Youtube 1
Youtube 2
Sleeping
mat 120 exam
Work
Bicycling
Sleeping
Schedule
Contacts
Chats
Profile


9:36
Fixed
AI Generated
Activity Title
Priority
Start Date
Start Time
End Date
End Time
Category
Routine
Privacy
Friends
Members
Manage Participants
RA
Rasul Aliyev
Add Activity


9:37
Fixed
AI Generated
Activity Title
Test title
Priority
June 15, 2020
15:37
Jun 19, 2020
21:35
Duration
00
55
01 hour
0 min.
02
5
Category
Academic
Privacy
Private
Members
Manage Participants
RA
Rasul Aliyev
Add Activity


9:40
Schedule
June
15 - 21
Mon
Tue
Wed
Thu
Fri
Sat
Sun
10:00
11:00
12:00
13:00
14:00
15:00
16:00
17:00
18:00
19:00
20:00
21:00
22:00
23:00
mat 120 exam
Work
Signal Exam
Academic
Jun 17, 2020
100
13:00 - 18:00
Prepare for Signal
Routine
Jun 16, 2020
19:00 - 22:00
10
Schedule
Contacts
Chats
Profile
9:38
Schedule
10
Youtube1
00:28 - 01:28
Jun 15, 2020
70
Sleeping
02:30 - 10:30
Jun 15, 2020
54
Youtube2
00:30 - 01:30
Jun 16, 2020
10
Prepare for Signal
19:00 - 22:00
Jun 16, 2020
100
Signal Exam
13:00 - 18:00
Jun 17, 2020
80
Zoom meeting
16:00 - 16:30
Jun 17, 2020
56
mat 120 exam
09:39 - 12:39
Jun 18, 2020
39
Work
10:30 - 12:30
Jun 19, 2020
10
Bicycling
13:00 - 16:00
Jun 19, 2020
10
Graduation project
19:00 - 23:00
Jun 19, 2020
38
Sleeping
03:00 - 10:00
Jun 20, 2020
Schedule
Contacts
Chats
Profile


9:41
Activity
Signal Exam
Privacy:
Personal
Category:
Academic
Date:
Jun 17, 2020
Start Time:
13:00
End Time:
18:00
Owner:
RA
Rasul Aliyev
Delete Activity

9:38

Search

Rustam Quliyev
@rustambaku13

Remove

Orkhan Salahov
@orkhan7887

Remove

My Groups

+ New Group

Schedule | Contacts | Chats | Profile

9:38

Search
rustam

Rustam Quliyev
@rustambaku13

Remove

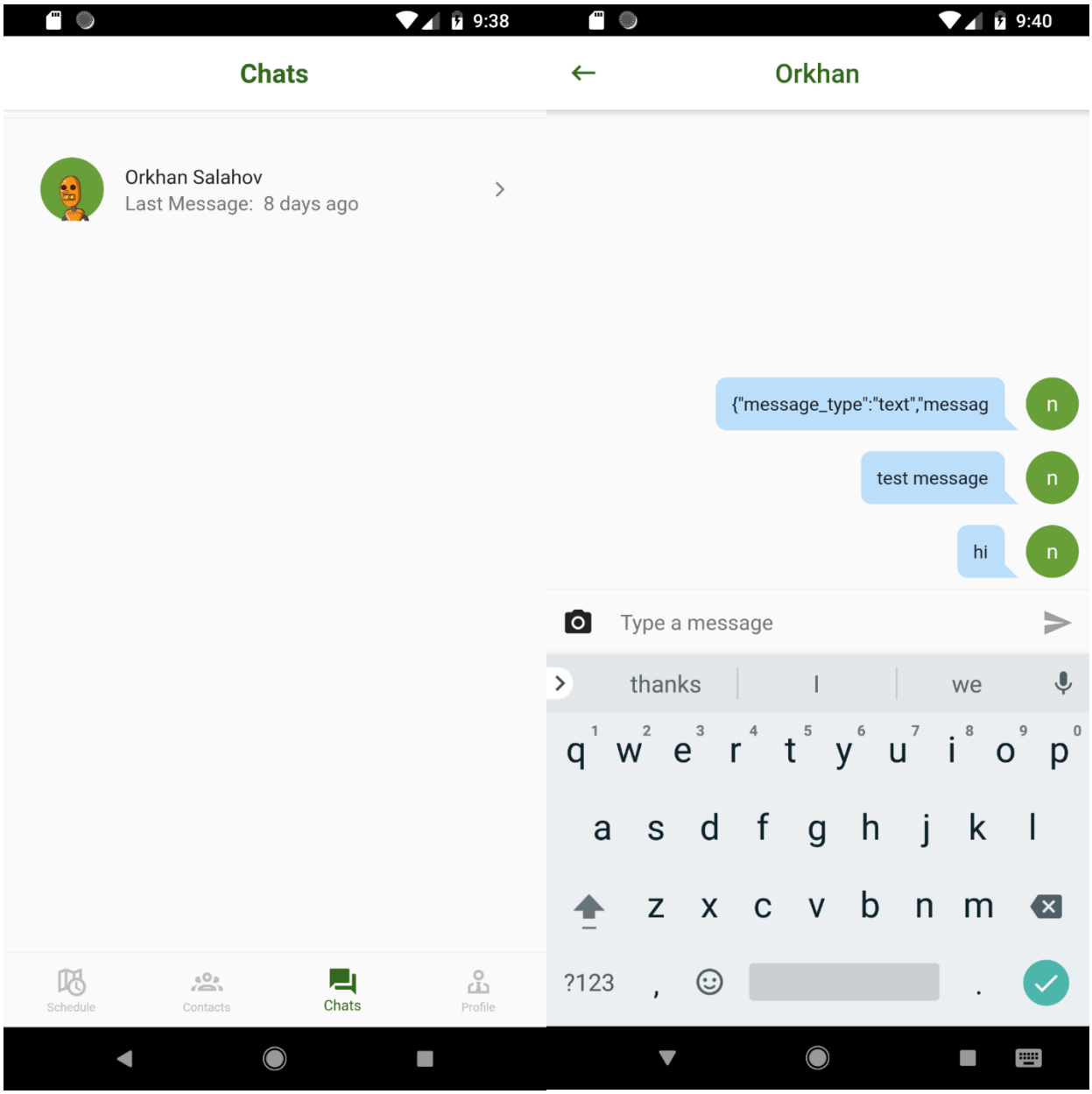
Naruto Uzumaki
@rustamnaruto

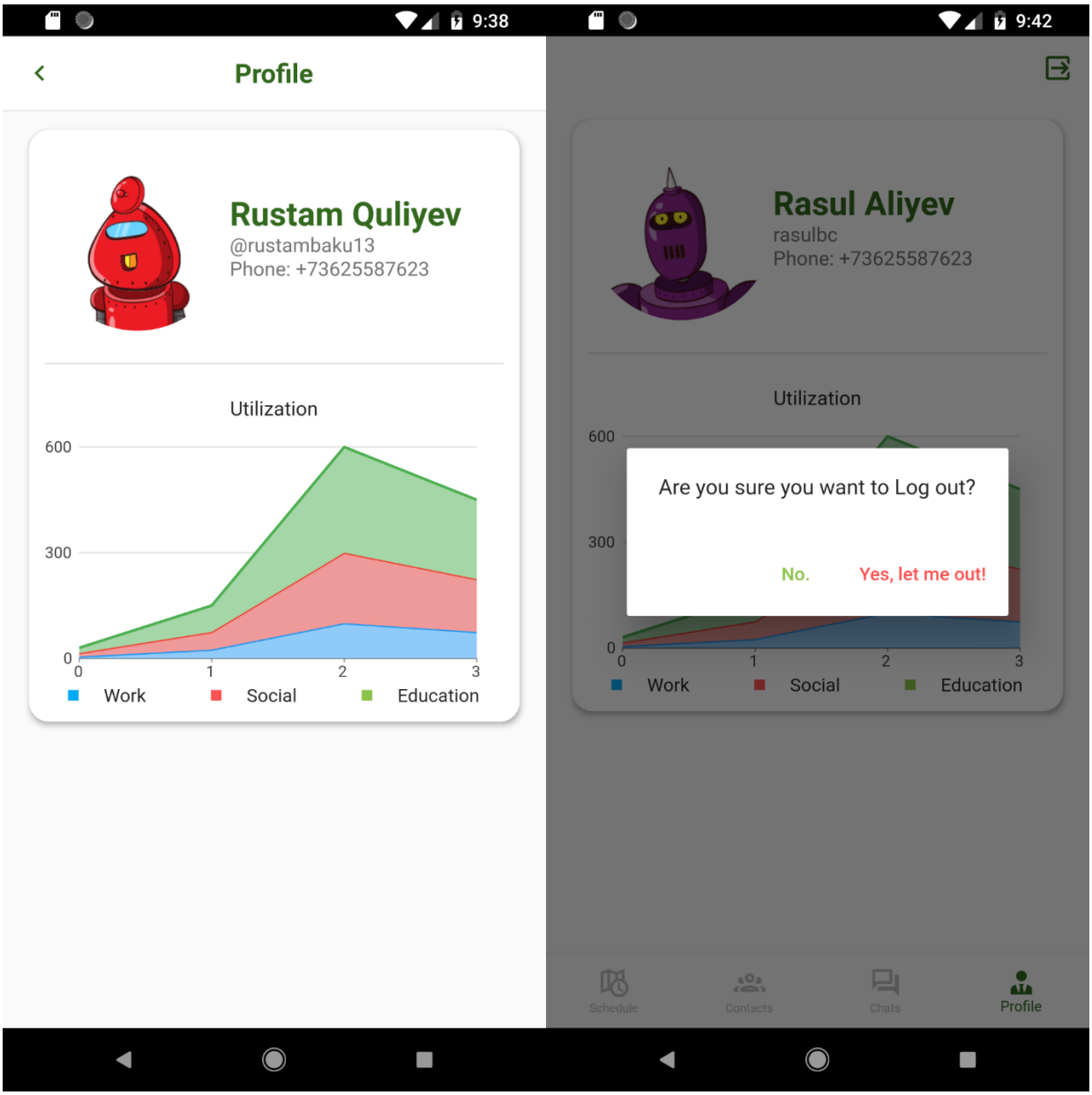
+ Add

My Groups

+ New Group

Schedule | Contacts | Chats | Profile


9:38
Chats
Orkhan Salahov
Last Message: 8 days ago
Schedule
Contacts
Chats
Profile
9:40
Orkhan
{"message_type":"text","messag
n
test message
n
hi
n
Type a message
thanks
I
we
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
?123
,
.


9:38
Profile
Rustam Quliyev
@rustambaku13
Phone: +73625587623
Utilization
600
300
0
0
1
2
3
Work
Social
Education
9:42
Rasul Aliyev
rasulbc
Phone: +73625587623
Utilization
600
300
0
0
1
2
3
Work
Social
Education
Are you sure you want to Log out?
No.
Yes, let me out!
Schedule
Contacts
Chats
Profile

4. Research Method & Design Details

3.1. Low Level Architecture Diagram

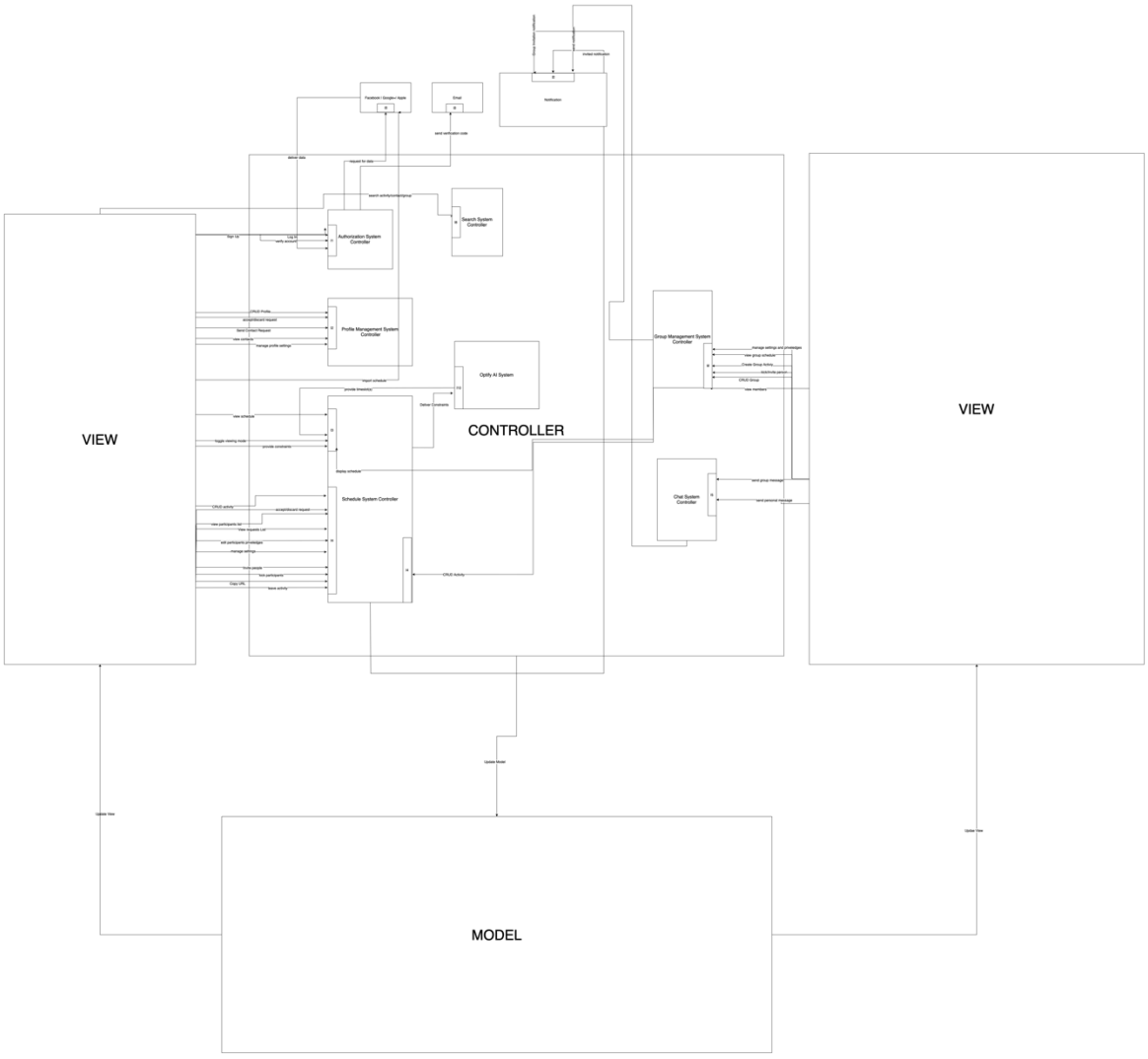


Figure 7 | Architecture Diagram MVC

https://drive.google.com/file/d/1obZJjcNZmB9VfUg0BErrmrNVQIBlhghz/view?usp=sharing

3.2. ERD Diagram

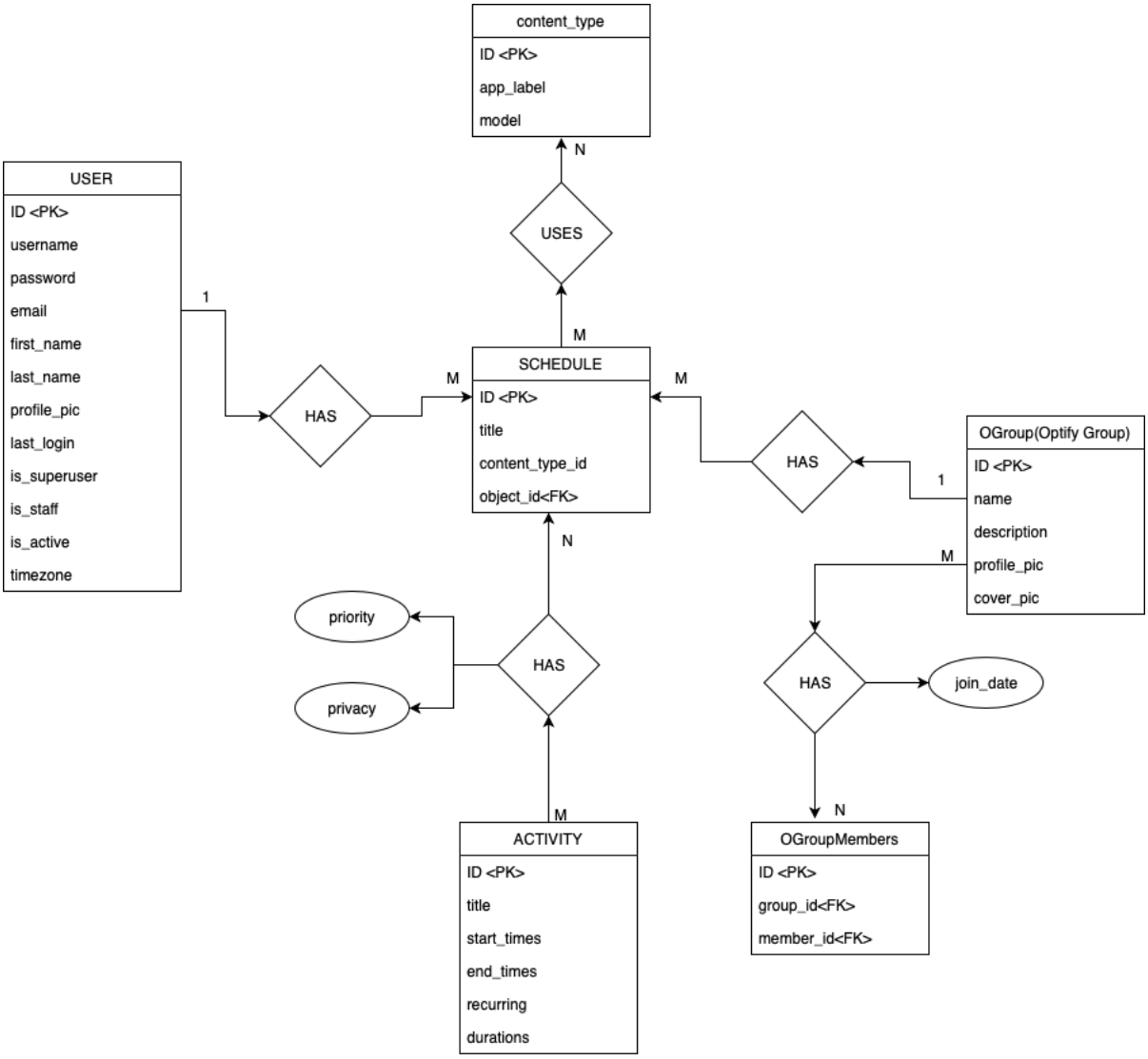


Figure 8 | ERD Diagram

3.3. Process Data Flow Diagram for activity Creation

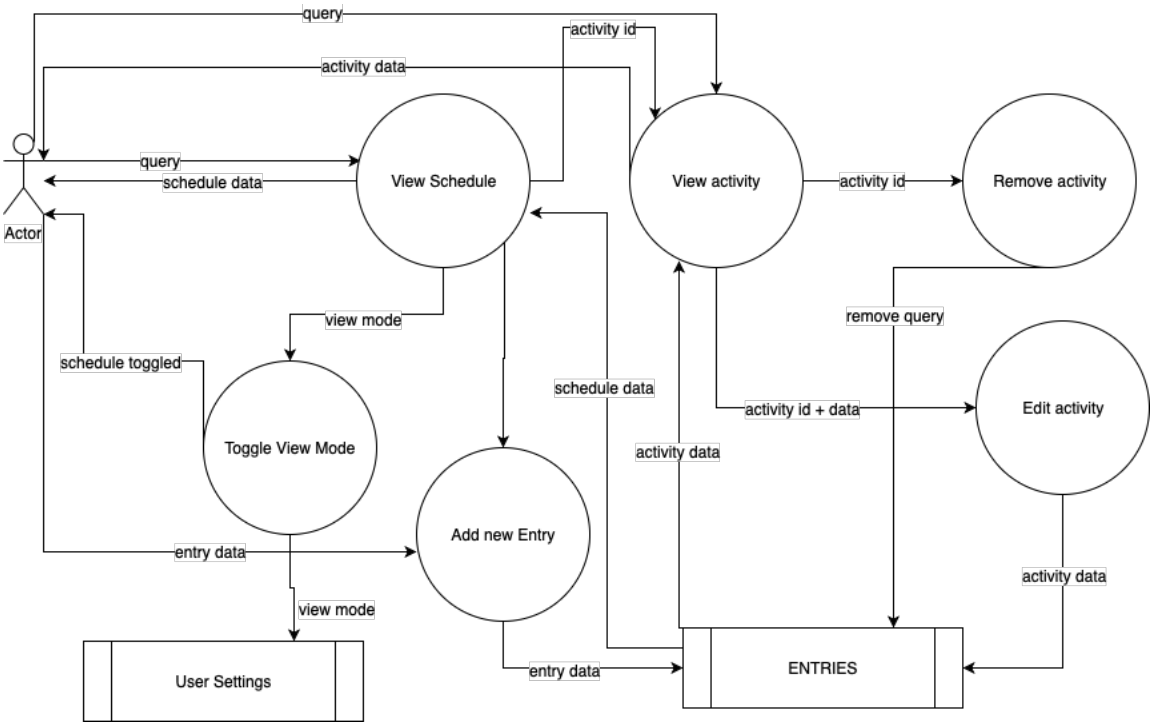


Figure 9 | DFD Level-1

3.4. Sequence Diagram for live activity Creation

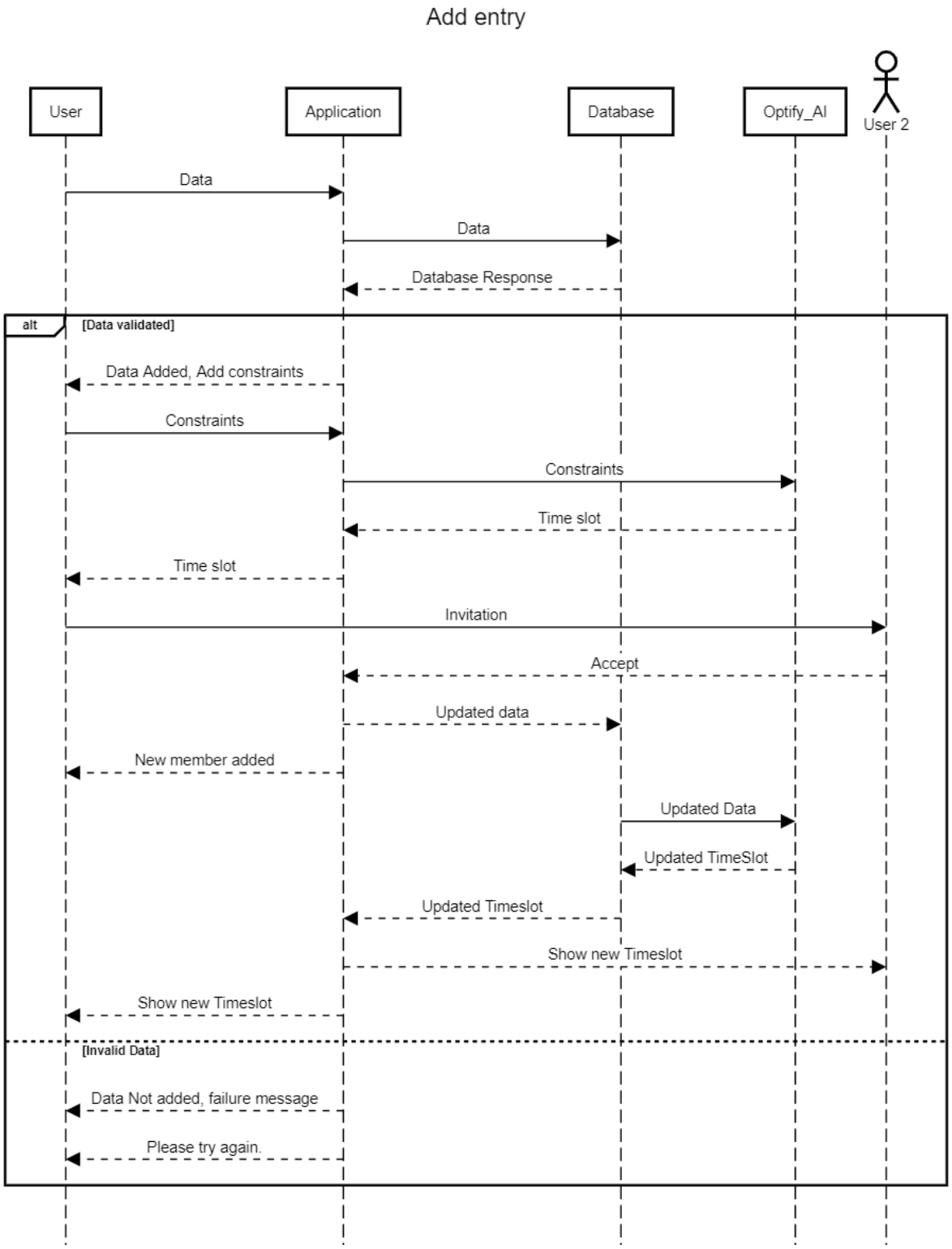


Figure 10 | Add Live Entry and Invite a person Sequence Diagram

3.5. Algorithms

The main algorithms that are heavily used in the project are for the AI part of the problem. The proposed solution comprises 2 parts:

- Local search with Gradient Descent
- Neural Networks (LSTM,Dense,Conv2D, Attention mechanism and etc…)

Local Search with Gradient Decent

This was part of our solution for the previous semester project, however for this semester we did a great upgrade on it. First of all we have studied **PyTorch** library which allowed us to transfer the entire pure python calculations into PyTorch. The motivation behind it was very simple, since we are using Gradient Decent to find the optimum of a function we need a method to calculate the **gradient**. Previous semester we were calculating gradient by doing a simple numerical approximation.

$$\nabla\phi(x) = \frac{\phi(x + \Delta t) - \phi(x)}{\Delta t}$$

By taking Delta of T sufficiently we were approximating our gradient function. This was doing very poor in performance since we had to calculate the gradient for every small step. In contrast, PyTorch has a built-in **autograd** module, which computes the gradients analytically as we apply the function. It also has built in **SGD**(Stochastic Gradient Decent), which is more advanced as to what we have built.

The idea behind this Local Search approach is quite simple, and intuitive. If we assume that the best activity timeslot is one that maximizs the distance between all activities, we can create a function that mimics this assumption.

Step 0

Given List of other activities

My variable: **Start Time** of the activity

Goal : Find the best Start Time such that the function defined below is minimum

Step 1

For all the activities that do not overlap with mine calculate the function below.

$$z=min\left(1,l\cdot e^{-l\left(duration-2\cdot sqrt\left(distance\right)\right)}\right)$$

Figure 11 | Function 1

Where duration is the duration of the activity that I am comparing to and distance is the distance between my Start Time and that activity(In Hours). Here is a 3D view of the function.

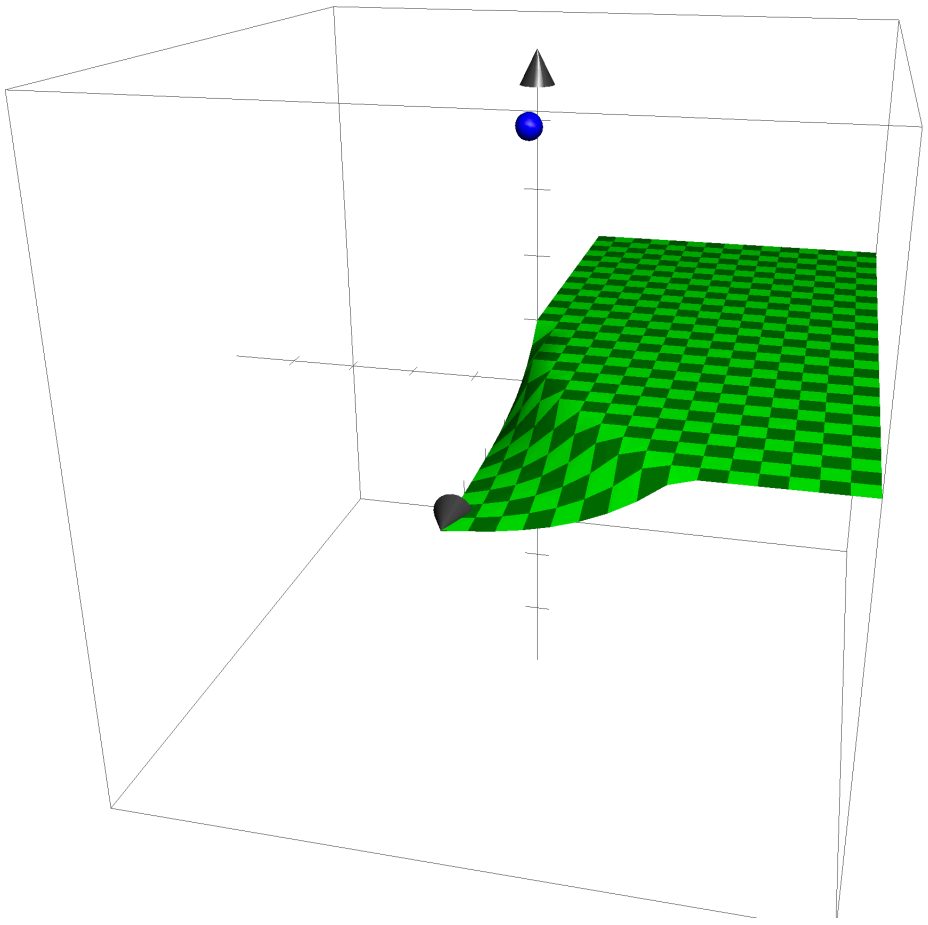


Figure 12 | 3d Function View

The intuition is simple, the larger the distance between My **Start Time** and Activity the more of its length I can tolerate.

Suppose the distance is 0.5 hours (a.k.a 30 minutes). Here is how the graph cross section would look like.

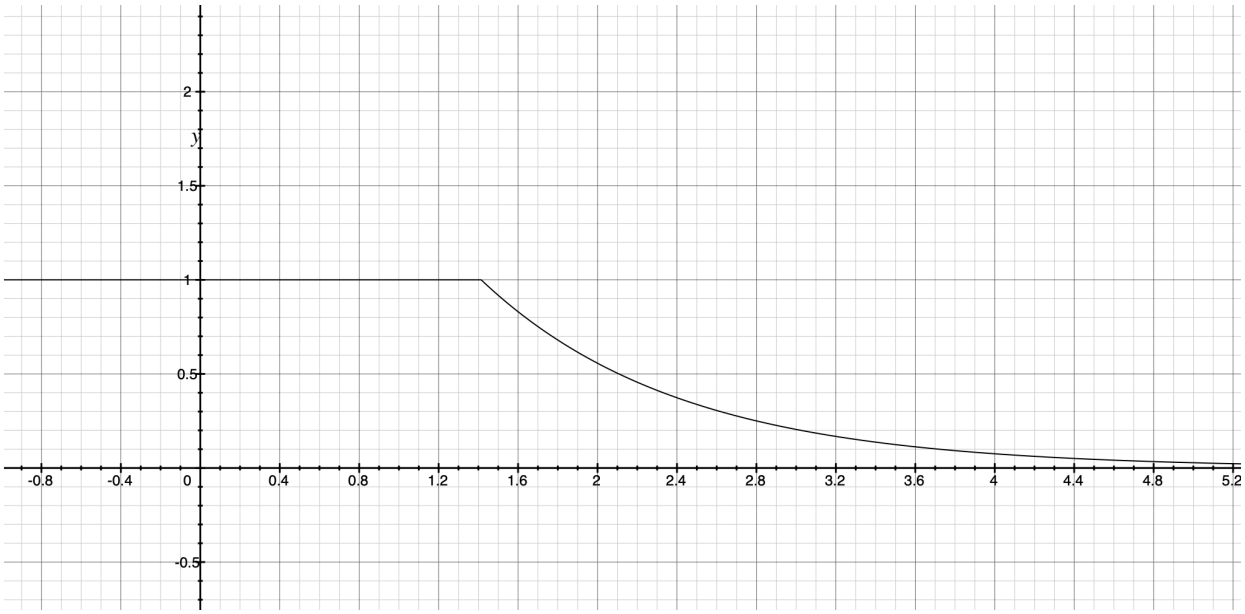


Figure 13 | Graph 2d

We can see that for Duration of 1.6 hours I can tolerate (selection probability is 1), thereafter the probability is rapidly dropping.

Step 2

For all the activities that overlap calculate the function below.

$$y=0.5\cdot e^{-5overlap}$$

Figure 14 | Function 2

Notice the input is the percent of overlap [0,1].

Figure 15 | Function 2 2D Graph

If 2 activities are barely overlapping probability of me choosing the timeslot is **0.5**. It is exponentially dropping until it reaches 1 where it is going to be **0**.

Step 3

For all activities calculate the function in **Step 1** if activities are non-overlapping, otherwise calculate **Step 2**.

Step 4

Return the -Log(answer from **Step 3**) in order to make the function negatively sloped so we can start descending. Below is a brief code describing this process. Afterwards we can simply input this function to SGD of Pytorch where it finds the best **Start Time.** Optionally assuming the activity is better to be at the center of a day we can multiply everything to a Normal Distribution of the following shape. Where **mean = 960** and **std = 400**. Variables are minutes from the start of the day.

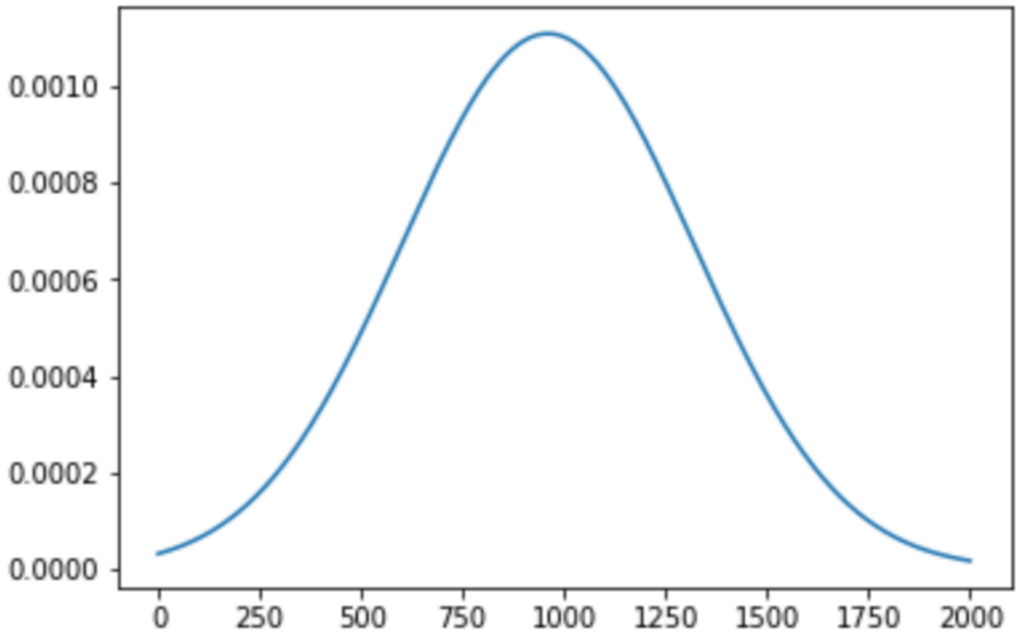


Figure 16 | Normal Function for a Day

```
def distanceLoss(starts,ends,start_time,duration,norm = Normal(torch.tensor([960.]),
torch.tensor([360.]))):
    """ Function is going to be minimized should be the loss function. Probability of
me liking a timeslot based on other distances. Multiplied by -(1) """
    distances = []
#     count = len(starts)
    for ind,start in enumerate(starts):
            if ends[ind] < start_time:
                # I am after this guy
                dist = (start_time - ends[ind])/60
                dur = (ends[ind] - start)/60

                distances.append(td_exponential_inverse(dur,dist))

            elif start > start_time+duration:
                #I am before this guy
                dist = (start - start_time - duration)/60
                dur = (ends[ind] - start)/60
                distances.append(td_exponential_inverse(dur,dist))
            else:
                o_s = torch.max(start_time,start)
                o_e = torch.min(start_time + duration,ends[ind])
                overlap_percent = ((o_e-o_s))/duration
                distances.append(exponential_5(overlap_percent))
```

```
    kk = torch.cat(distances,dim=0)
    product = torch.prod(kk)
    norm = torch.exp(norm.log_prob(start_time))
#     print("Product",product.item(),"Norm",norm.item())
    return -torch.log(product * norm)
```

In Action

Here is our heuristic approach in practice, given a calendar with following entries I want to add a new activity for 50 minutes within that day.

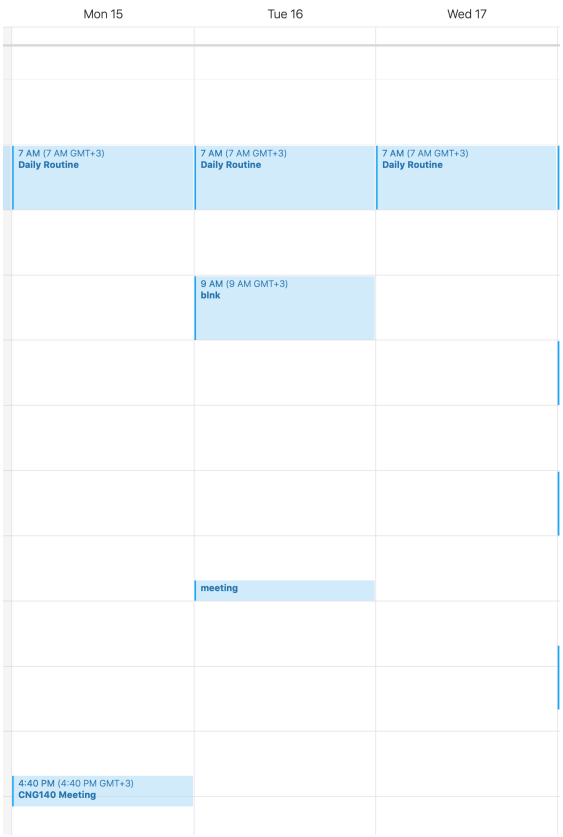
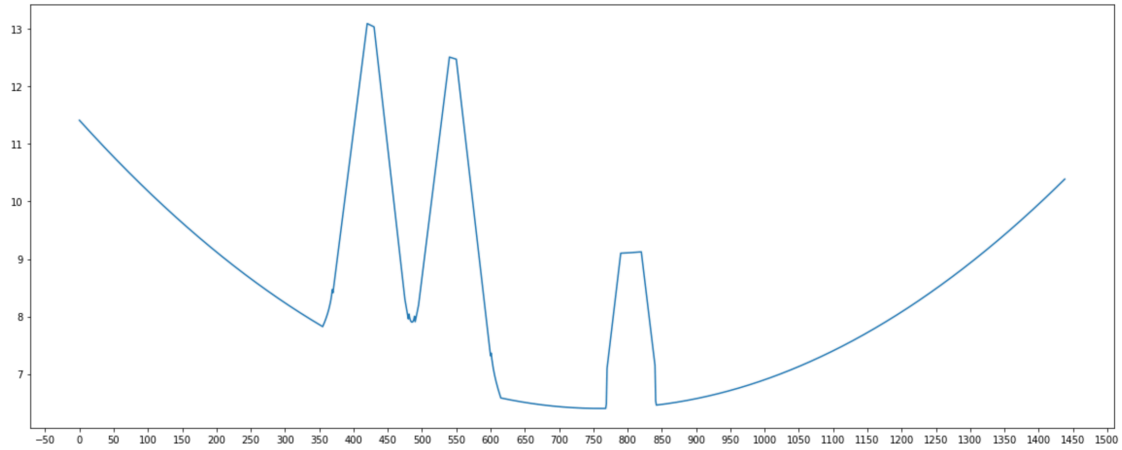


Figure 17 | Example Schedule

```
[485.4111633300781, 844.154296875, 653.3472900390625] [7.900360584259033, 6.461058616638184, 6.498527526855469]
1 Is chosen as Argmin
844.154296875
```



We can see that those high peaks appear when we have overlaps. And we can see that the most optimal timeslot has been found as **844**, which is exactly after our small meeting.

ANN (Artificial Neural Network)

As an addition to the previous heuristic solution, users' schedules and their preferences differ from user to user. After extensive research we can to a conclusion that the best solution would be to create a central Neural Network that is going to analyze users' data and make suggestions.

Inputs:

- Activity's Title
- Activity's Duration
- User's Id
- Other Activities within the range of +-1 day's titles
- Timetable array
- Activity's Titles' Characters
- Other Activities within the range of +- 1 day's characters

Outputs:

- 24 length Vector of probabilities for each hour in a day

Architecture of the Model

Our architecture has sever main components.

1. Intention Layer

In this layer the goal is calculate the intention of the activity.

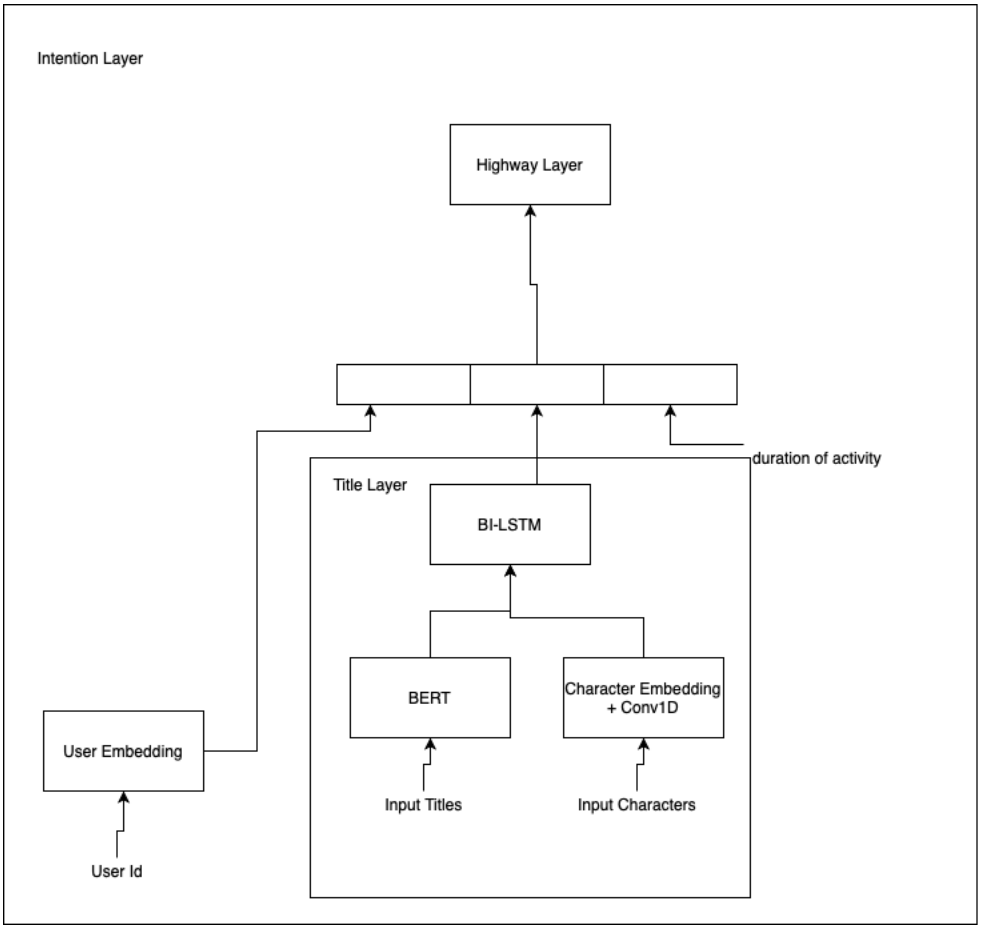


Figure 18 | intention Layer

2. Other Activities Layer

For all the other activities in the context we are calculating the context as defined in the component above.

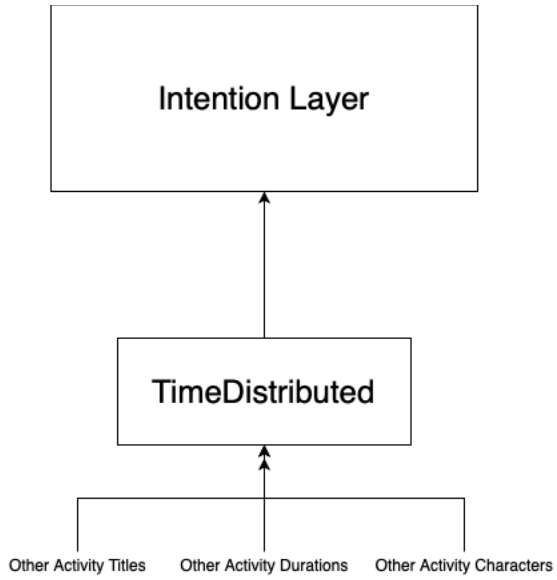


Figure 19 | Other Activities Layer

3. TimeTable Layer

A time table input is defined as matrix of size (3*24*12,6). Which means that for 3 days if we divide each hour into 12 parts (5 minute each) and allow each timeslot to have at most 6 overlapping activities. This timetable represents the `image` of our current schedule in a given context (+- 1 day) .

Here is an example timetable reshaped into viewable format. Yellow regions denote more overlapping activities.

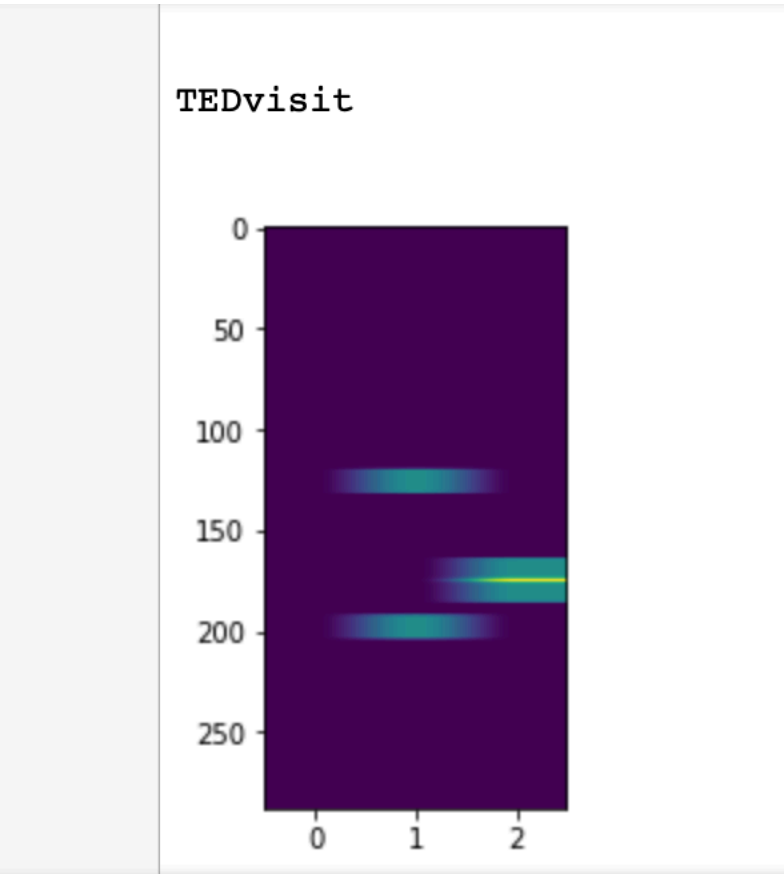


Figure 20 | Time Table Example

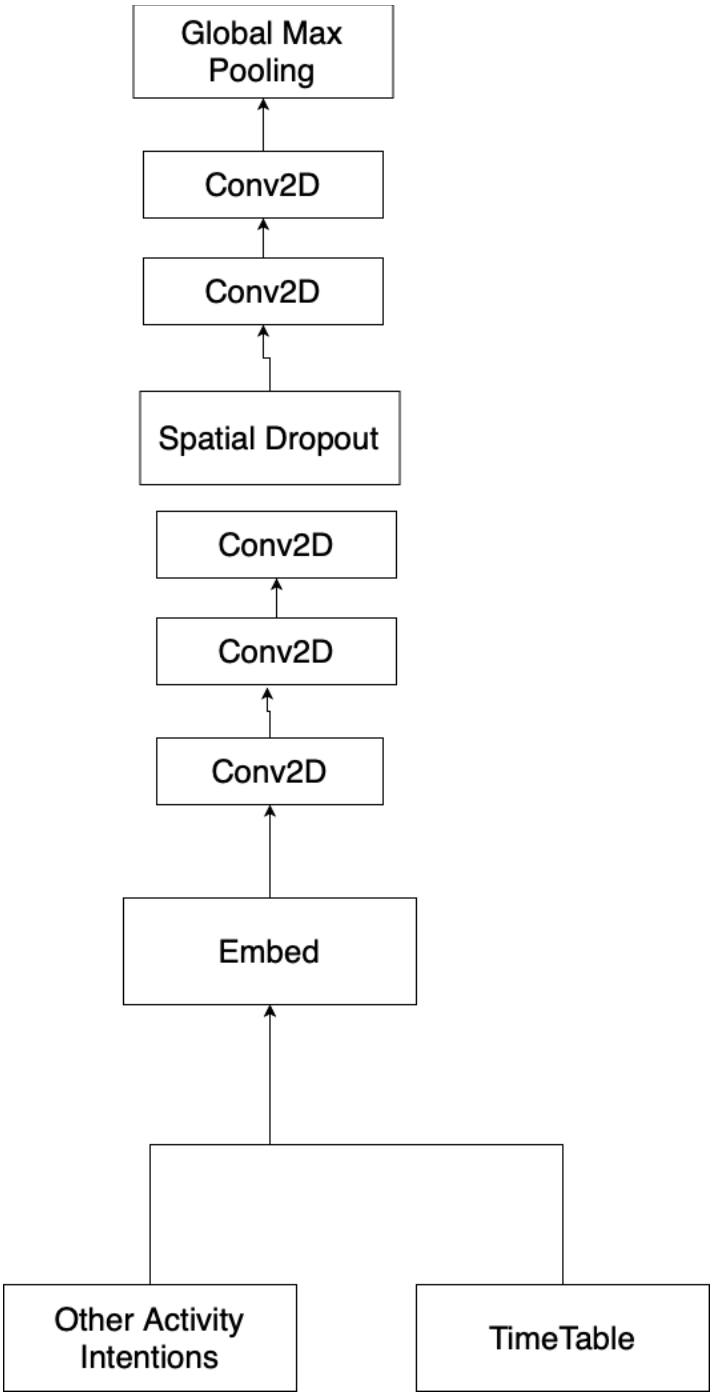


Figure 21 | TimeTable Layer

4. Final Layer

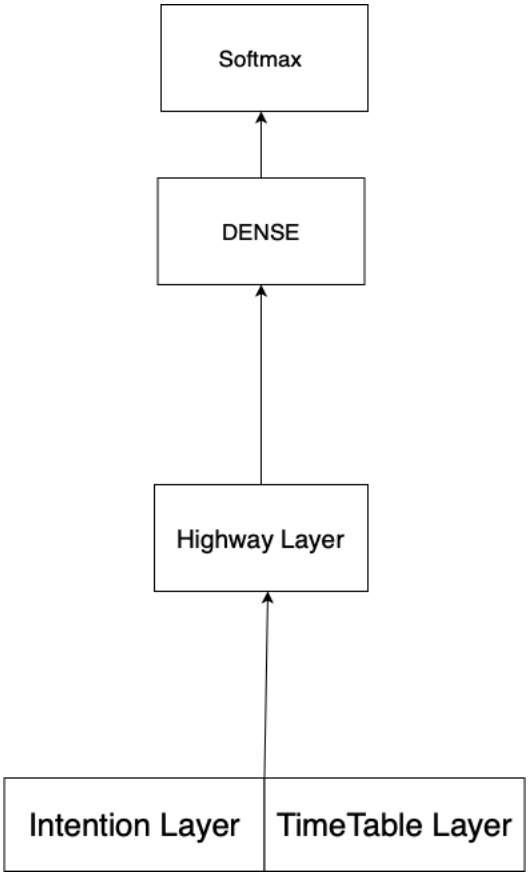


Figure 22 | Softmax Layer

Summary

```
Layer (type)                    Output Shape         Param #     Connected
to
==================================================================================
========================
user_id (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________
__________________________
othcer_act_words (InputLayer)   [(None, 30, 64)]     0
__________________________________________________________________________________
__________________________
other_act_characters (InputLaye [(None, 30, 64, 64)] 0
__________________________________________________________________________________
__________________________
```

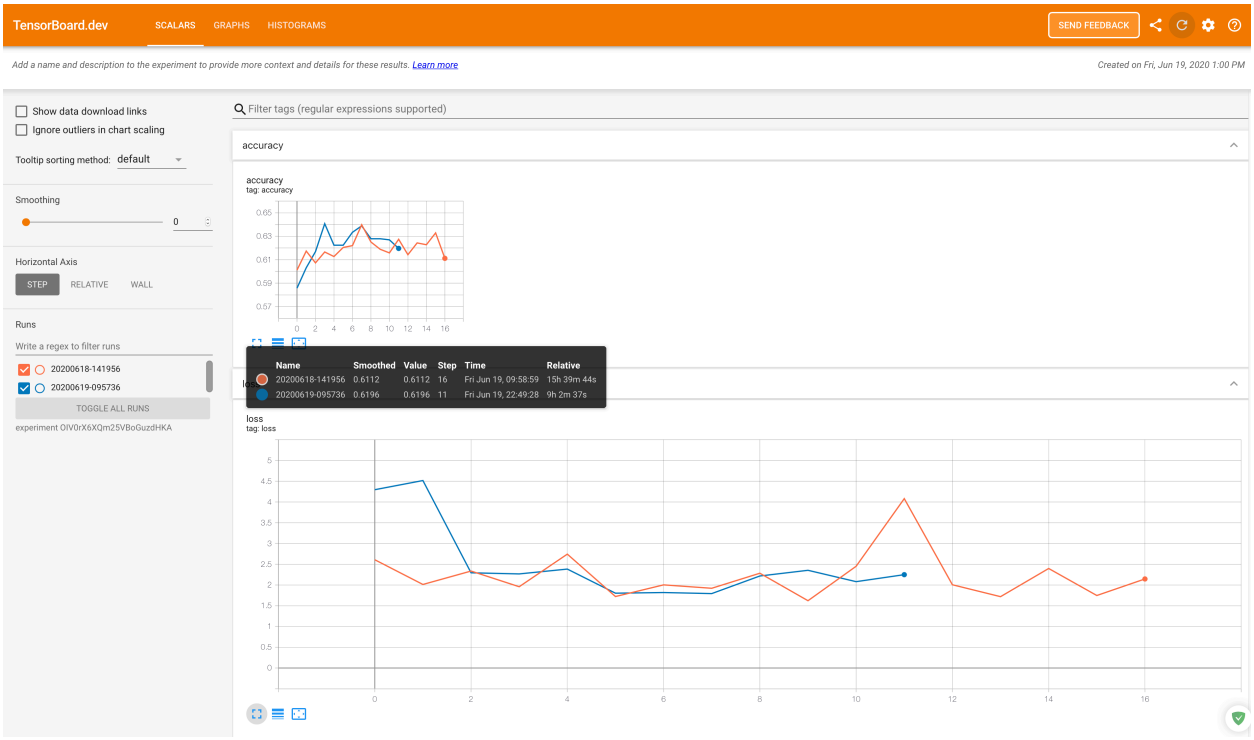
```
embedding_3 (Embedding)         (None, 1, 256)       2560        user_id[0
][0]
__________________________________________________________________________
_____________________
time_distributed_3 (TimeDistrib (None, 30, 64, 768)  178444032   othcer_ac
t_words[0][0]
__________________________________________________________________________
_____________________
time_distributed_2 (TimeDistrib (None, 30, 64, 768)  276224      other_act
_characters[0][0]
__________________________________________________________________________
_____________________
reshape (Reshape)               (None, 256)          0           embedding
_3[0][0]
__________________________________________________________________________
_____________________
concatenate_2 (Concatenate)     (None, 30, 64, 1536) 0           time_dist
ributed_3[0][0]
                                                                 time_dist
ributed_2[0][0]
__________________________________________________________________________
_____________________
time_distributed_4 (TimeDistrib (None, 30, 256)      1704960     concatena
te_2[0][0]
__________________________________________________________________________
_____________________
other_durations (InputLayer)    [(None, 30, 1)]      0
__________________________________________________________________________
_____________________
repeat_vector (RepeatVector)    (None, 30, 256)      0           reshape[0
][0]
__________________________________________________________________________
_____________________
concatenate_3 (Concatenate)     (None, 30, 513)      0           time_dist
ributed_4[0][0]
                                                                 other_dur
ations[0][0]
                                                                 repeat_ve
ctor[0][0]
__________________________________________________________________________
_____________________
time_distributed_5 (TimeDistrib (None, 30, 513)      527364      concatena
te_3[0][0]
__________________________________________________________________________
_____________________
zero_padding1d (ZeroPadding1D)  (None, 31, 513)      0           time_dist
ributed_5[0][0]
__________________________________________________________________________
_____________________
input_words_ids (InputLayer)    [(None, 64)]         0
__________________________________________________________________________
_____________________
input_words_characters (InputLa [(None, 64, 64)]     0
```

```
__________________________________________________________________________________________________
tf_op_layer_strided_slice (Tens [(31, 513)]          0           zero_padding1d[0][0]
__________________________________________________________________________________________________
timetable (InputLayer)          [(None, 864, 6)]     0
__________________________________________________________________________________________________
bert_wrapper_1 (BertWrapper)    (None, 64, 768)      178444032   input_words_ids[0][0]
__________________________________________________________________________________________________
time_distributed (TimeDistribut (None, 64, 768)      276224      input_words_characters[0][0]
__________________________________________________________________________________________________
tf_op_layer_GatherV2 (TensorFlo [(None, 864, 6, 513) 0           tf_op_layer_strided_slice[0][0]
                                                                 timetable[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 64, 1536)     0           bert_wrapper_1[0][0]
                                                                 time_distributed[0][0]
__________________________________________________________________________________________________
tf_op_layer_Identity (TensorFlo [(None, 864, 6, 513) 0           tf_op_layer_GatherV2[0][0]
__________________________________________________________________________________________________
sequential_10 (Sequential)      (None, 256)          1704960     concatenate[0][0]
__________________________________________________________________________________________________
duration_input_hours (InputLaye [(None, 1)]          0
__________________________________________________________________________________________________
tf_op_layer_Transpose (TensorFl [(None, 864, 513, 6) 0           tf_op_layer_Identity[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 513)          0           sequential_10[0][0]
                                                                 duration_input_hours[0][0]
                                                                 reshape[0][0]
__________________________________________________________________________________________________
sequential_11 (Sequential)      (None, 864, 513, 300 1682400     tf_op_layer_Transpose[0][0]
```

```
_________________________________________________________________
highway_layer_2 (HighwayLayer)  (None, 513)          527364      concatena
te_1[0][0]
_________________________________________________________________
sequential_12 (Sequential)      (None, 150)          435900      sequentia
l_11[0][0]
_________________________________________________________________
weekday_of_activity (InputLayer [(None, 7)]          0
_________________________________________________________________
concatenate_4 (Concatenate)     (None, 670)          0           highway_l
ayer_2[0][0]
                                                                 sequentia
l_12[0][0]
                                                                 weekday_o
f_activity[0][0]
_________________________________________________________________
highway_layer_3 (HighwayLayer)  (None, 663)          889746      concatena
te_4[0][0]
_________________________________________________________________
sequential_8 (Sequential)       (None, 24)           352280      highway_l
ayer_3[0][0]
=================================================================
Total params: 184,315,466
Trainable params: 6,461,126
Non-trainable params: 177,854,340
```

Data

The only bottleneck in our model was the lack of data. Unfortunately calendar data was not available in the web. Fortunately though Yeliz Hoca has provided us with her personal calendar data. However since the calendar was biased and our model was quite large we have faced difficulties training the model. Below you can see our training information with metrics of **Top5CategoricalAccuracy.** There were 2 trainings below. Blue one which incorporates activities **weekday** information and red one which does not. Because of such great number of parameters even for such a few epochs it took about **15** hours to train the network.



3.6. Data Structures

Trees, arrays ,dictionaries, graphs are heavily utilized within the project.

5. Implementation Details

5.1. Project Management

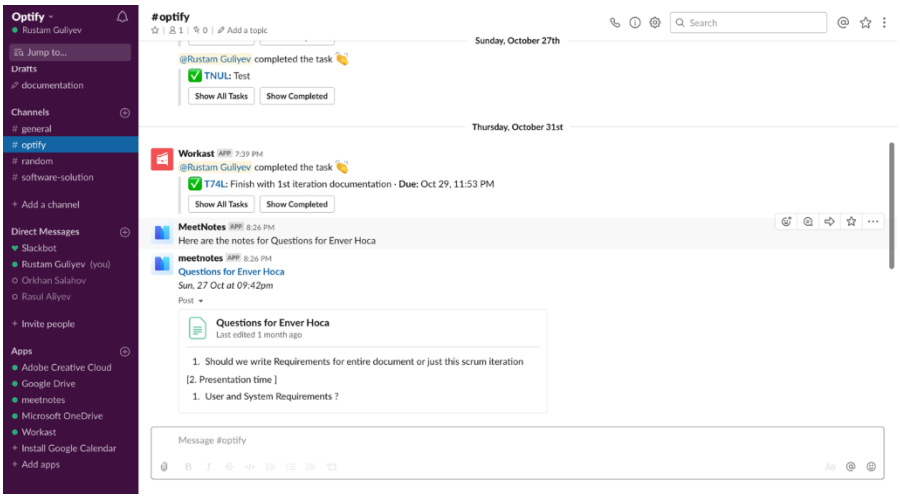
In order to coordinate our development process we were in great need of some project management software and version control tools.

Since scrum leader was Rasul, project management performance was poor.

5.1.1. Slack

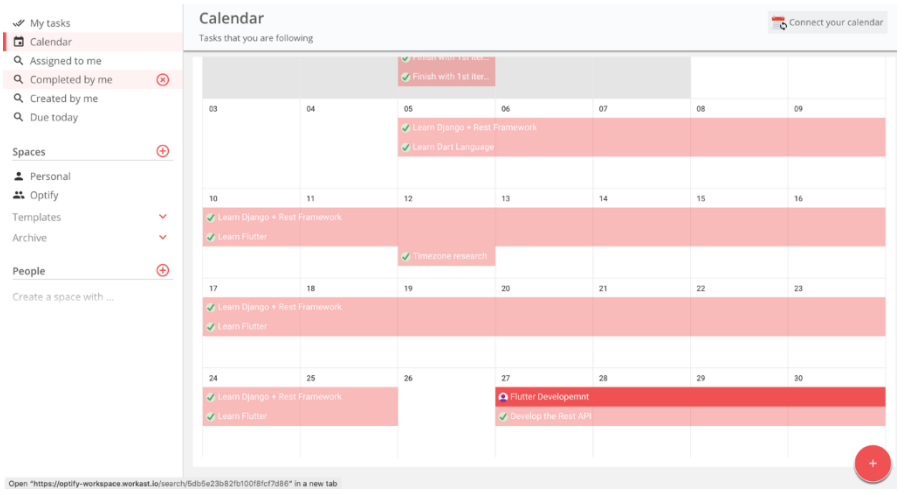
Slack is a project managements tool which is widely used in the field nowadays. We have had experience with it during my internship where most of the developers were chatting and planning using that tool. Also it has plenty of other 3-rd party applications within, which can be

added by a single click to the workspace. In Slack we have created our workspace at the beginning of the first iteration and were using some additional applications within as well.



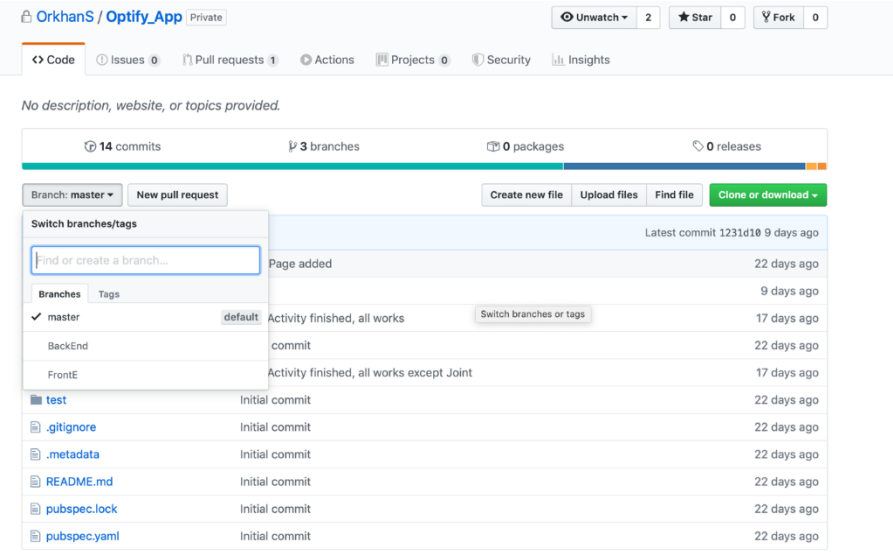
5.1.2. Workast

Workast is an application within slack that is useful for for project management. In Workast people can define tasks, deadlines and assign different people for the tasks. Due to Workast we have been tracking our Sprint Backlog and progress.



5.1.3. GitHub

GitHub is, arguably, the most popular version control tool in the market. We have been using it for quite some years now, and it delivers awesome performance. Also, thanks to student subscription we can have unlimited private repositories, hence no one can see our source code. We have created 2 separate branches for our project till now. FrontEnd and BackEnd branches. FrontEnd is currently being worked on Rasul and Orkhan who are actively developing separate sections of the app using Flutter and deploying to the branch. Occasionally I(Rustam) am pulling that branch to test the Application on Iphone simulators which are only available in Mac OSx. As for the BackEnd I am almost finishing the REST API using the Django Framework and actively deploying to the branch.



5.2. Testing

1. Unit Testing

Back-End Application

As it has been mentioned in the previous report, for the back end application testing we have used Soap UI framework.

Soap UI specializes in API testing where one can do the following tests.

- Functional Testing
- Performance Testing
- Security Testing
- Data Driven Testing
-

The figure below depicts the tree structure of the Soap UI Suit. At the very top we have our API resources, and endpoints. At the mid section we have Soap UI Test Suites. We have only 1 test suite(User Interaction) inside of which we have different test cases(Sign Up, Create Activity, Activities, Contacts, Groups). All of those test cases are composed of several test steps.

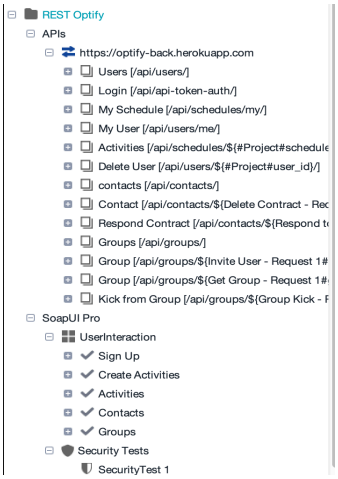


Figure 23 | Test structure

1. Sign Up Test Case.

- *Participants: Rustam*
- **Pass/Fail Criteria**: Create a new user, Register, and Delete. Make sure Schedule is allocated as soon as new user is created.

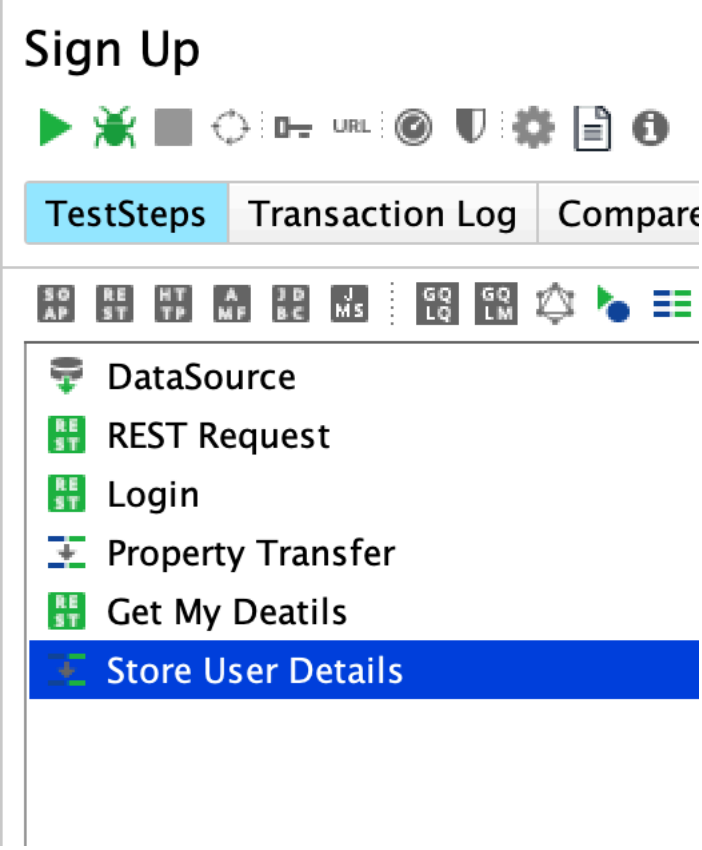


Figure 24 | Sign Up Test Case Setup

This are the steps in the Sign Up Test case.

1. Data Source is responsible for generating the data for new users' username and password
2. Rest Request is the actual user sign up request
3. During Login Request the users' authorization token is returned
4. Property Transfer is responsible for extracting that token from previous step and direct it to the next step
5. Finally we are getting current users details and storing the token globally so that requests afterwards can access it.

Below is the image of Data Source step, in more detail.

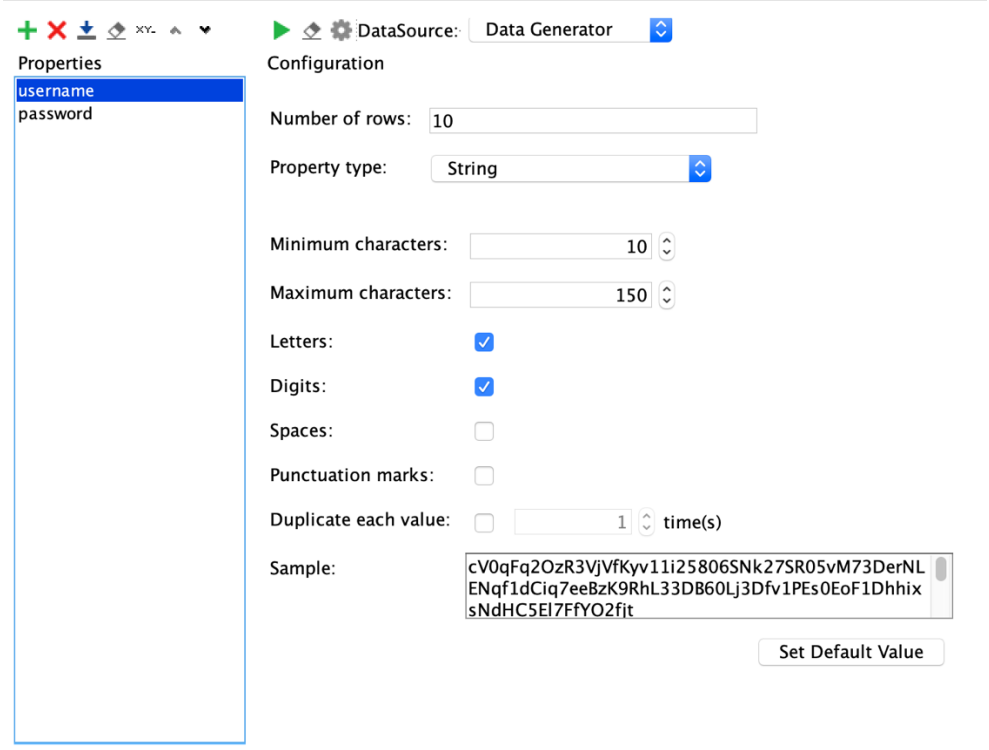


Figure 25 | Login Case Datastore

We can see exactly how username and password is being extracted from the data source.

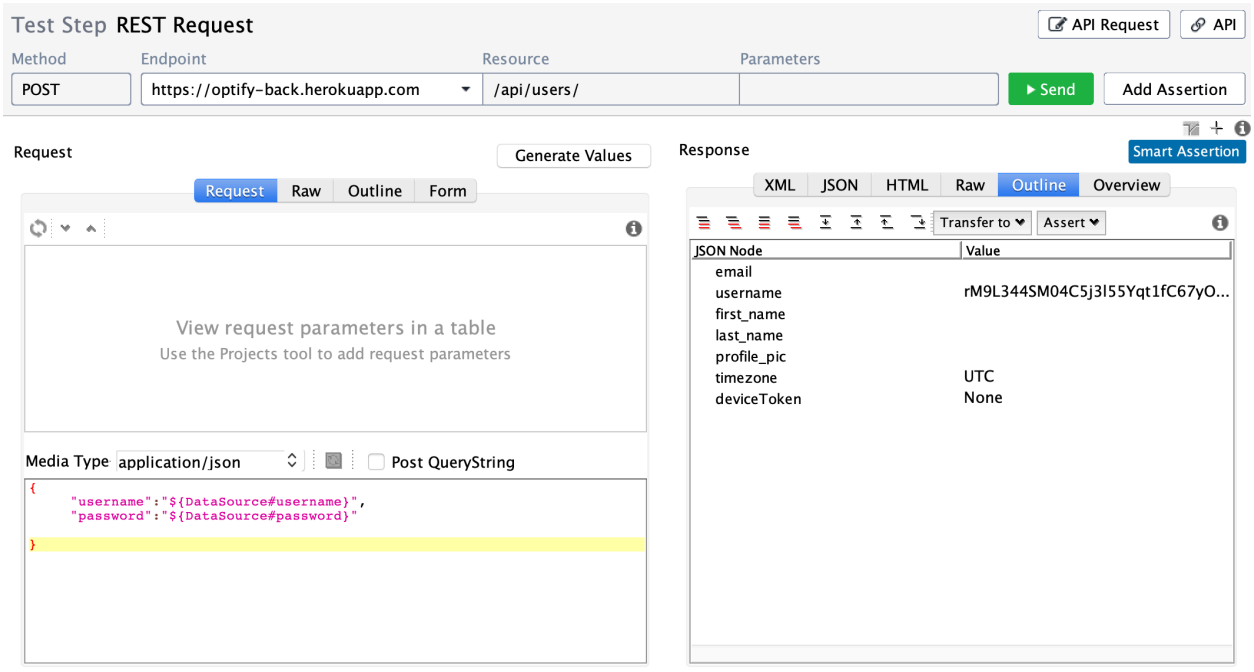


Figure 26 | Create user Request

For this Test Case the pass criteria was successfully executing all the instructions. Our server returns HTTP 200 for all successful GET requests and 201 for POST requests. Therefore our **assertion** was correct HTTP status code. Having a setup of such, we have initiated the test as a whole and everything went quite smooth. All the response codes were as expected.

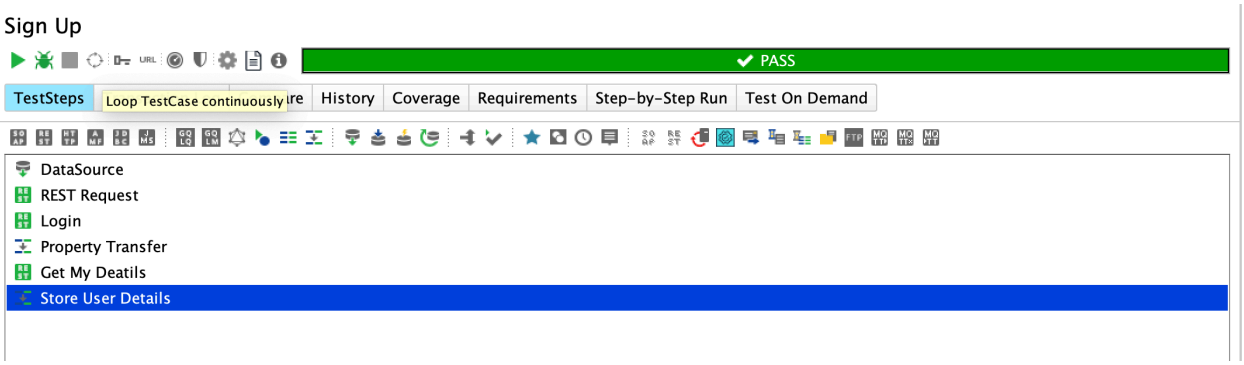


Figure 27 | Sign Up Test Case Results

2. Create Activities Test Case

- *Participants: Rustam*
- **Pass/Fail Criteria**: Correctly Fetch the Schedule, and create a new Activity within it.

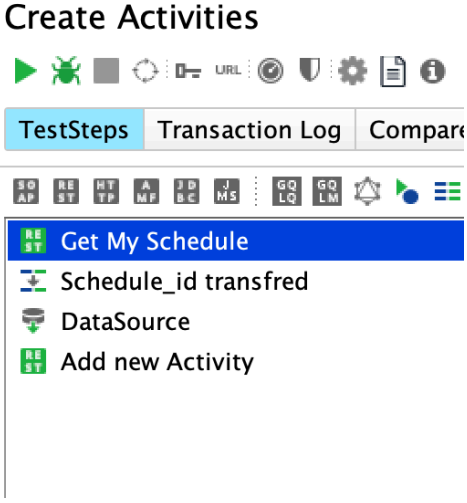


Figure 28 | Create Activities Test Case

Indeed this test case is quite similar to the first one.

1. Get Users Schedule Based on his token
2. Transfer users schedule id for next requests to use.
3. Generate fake data for start time, end time, title, priority of the activity
4. Create new activity from passed data

Below we can see Create Activity request, and how it uses the provided inputs from Data Source step.

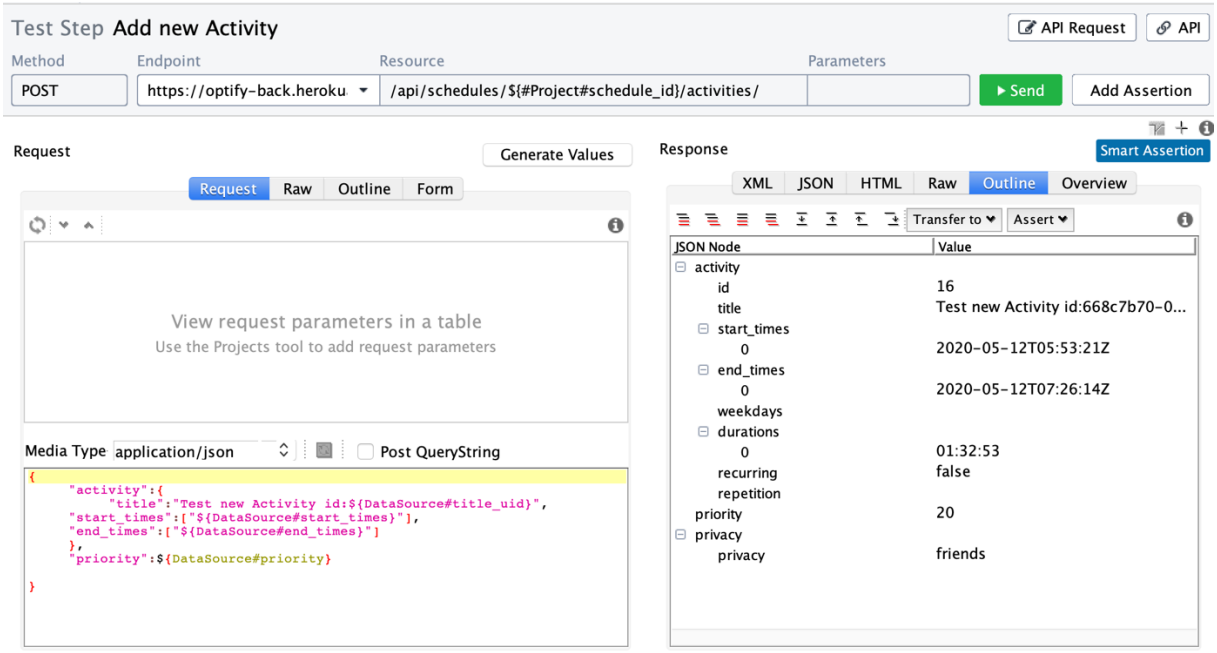


Figure 29 | Create Activity Test Case

Apart from request HTTP status assertions we are also checking the output JSON structure. As for example for getting users schedules we are asserting to have the owner field, since this is a field which is the most error prone.

Figure 30 | JSON Existance Assertion

In a similar way as in step 1 we have run all the test for this test case.

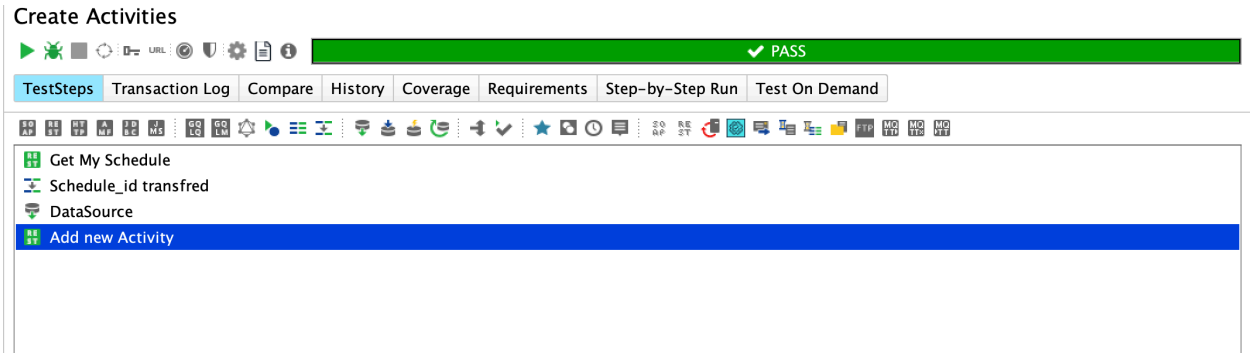


Figure 31 | Create Activites Test Case

We can see the newly created activity in JSON format by clicking on the last step in the pipeline.


```
{
   "activity" : {
      "id" : 120,
      "title" : "Test new Activity id:b0f61f15-99c2-497d-b9
      "start_times" : [
         "2020-05-12T05:52:12Z"
      ],
      "end_times" : [
         "2020-05-12T13:35:59Z"
      ],
      "weekdays" : null,
      "durations" : [
         "07:43:47"
      ],
      "recurring" : false,
      "repetition" : null
   },
   "priority" : 80,
   "privacy" : {
      "privacy" : "friends"
   }
}
```


Figure 32 | Created Activity

3. Activities View Test Case

- *Participants: Rustam*
- **Pass/Fail Criteria**: Display Serialized Multiple Activities in correct format

This is a rather simple test case which consists of 1 step, and that is Viewing the Schedule.

The assertions for this test step are checking the overall format of the JSON response.

We are checking so that it has 1 activity it, also we are making sure that all of the activities have *activity, priority & privacy* fields.

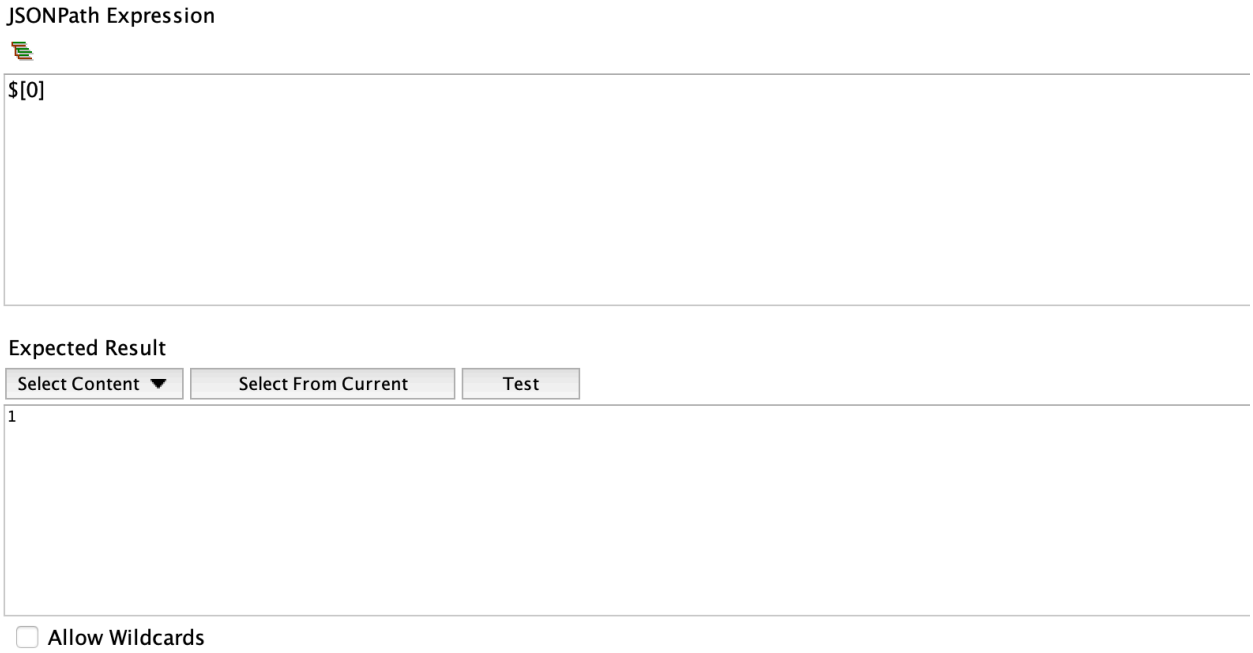


Figure 33 | Json count assertion

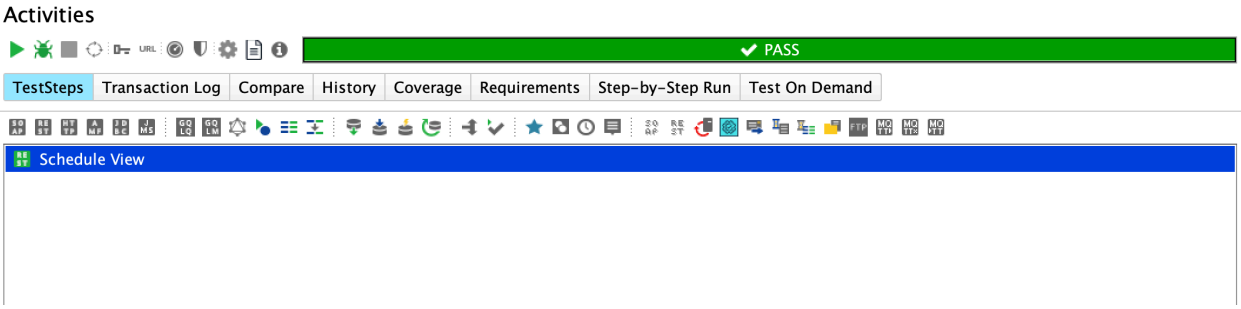


Figure 34 | Activity Test Step Run

4. Contacts View Test Case

- *Participants: Rustam*
- **Pass/Fail Criteria**: Create Contact request, Block duplicate requests, respond to the request correctly.

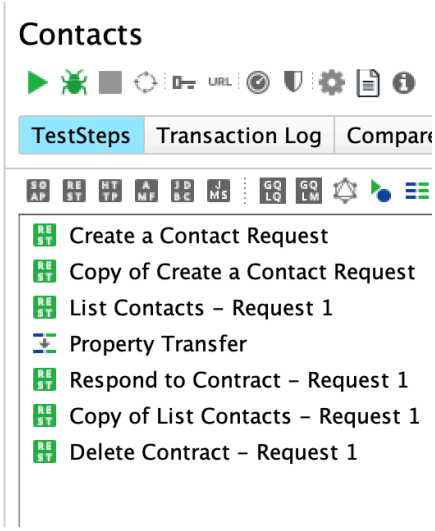


Figure 35 | Contacts Test Case

In this test case we are conducting the following.

1. Make a Contact Request to another user
2. Repeat the same request
3. List my current contacts
4. Accept the incoming request
5. List Contacts again
6. Delete from contact list

The repeating contact request is needed to make sure that the API blocks requests for second time, for that reason our assertion for the second step is not success HTTP codes but failed codes.

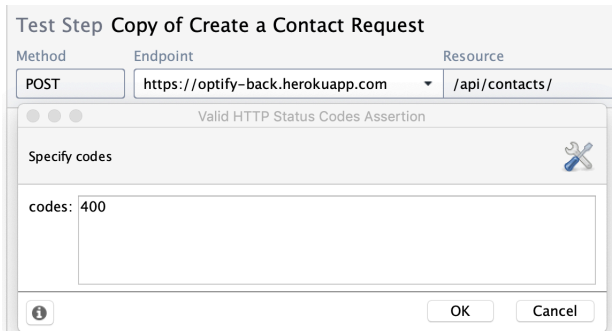


Figure 36 | Create Contanct repeat assertion

Initially when a new contact is created it is in **requesting** state, that is why in step 3(List my current contact) we are doing a JSON Match assertion to check the correctness of the state.

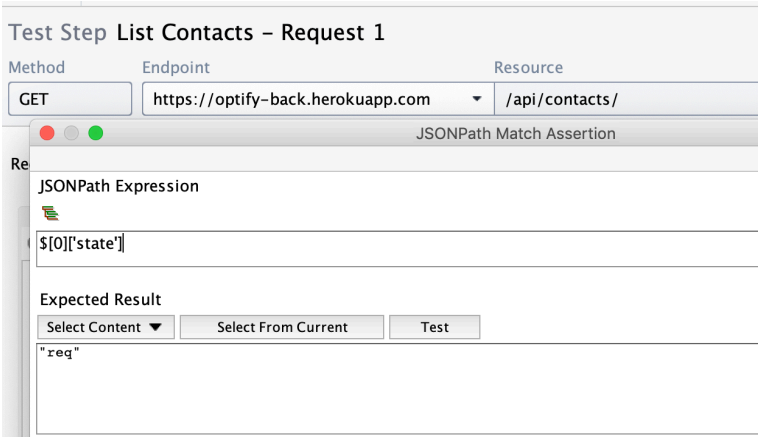


Figure 37 | List Contacts Request State

And after accepting the invitation our contacts state should become **accepted**. Hence the assertion below at step 5.

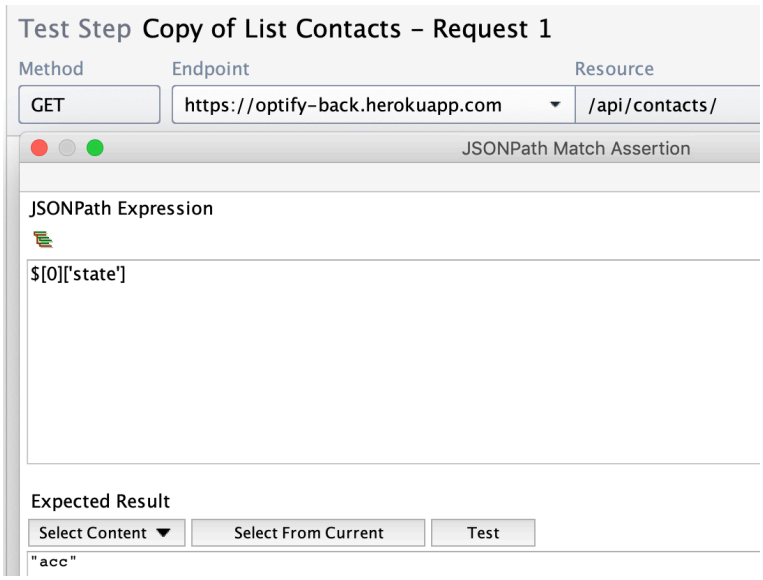


Figure 38 | List Contacts Request State 2

Lastly we are deleting our contact, and assertion for that is HTTP code 204. Below is one iteration of this test case.

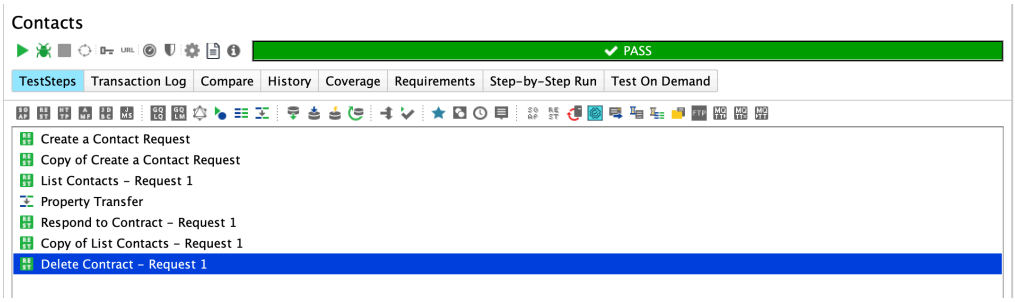


Figure 39 | List Contacts Test Case Run

We can indeed view the response at each step thus manually verifying that our states indeed do change after accepting.

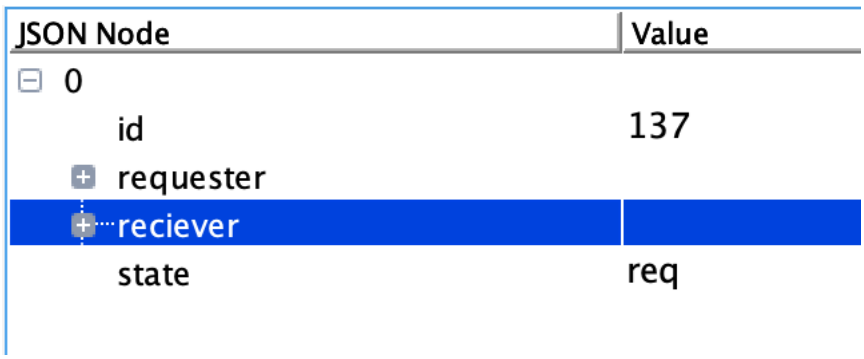


Figure 40 | List Contacts Test Run State 1

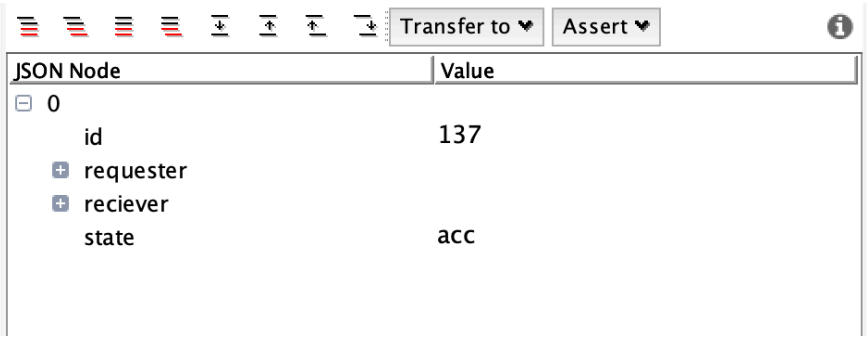


Figure 41 | List Contacts Test Run State2

5. Groups Test Case

- *Participants: Rustam*
- **Pass/Fail Criteria**: Create new Group, Add a new User, View Group, Kick the new User, Delete Group.

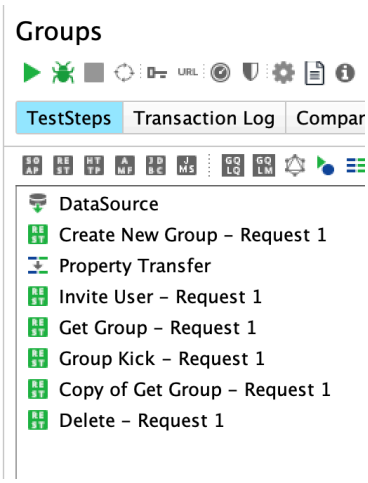


Figure 42 | Groups Test Case

1. Generate data for new groups
2. Create a new group with that data
3. Extract the group id for further queries
4. Add a User to the group
5. View group information
6. Kick User from group
7. View group information
8. Delete group

The generated data are the title and description of the new group.

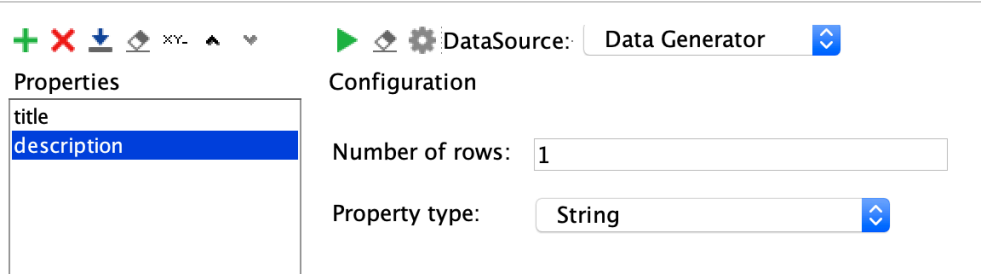


Figure 43 | Data Generator Group

For the 5th step(Viewing group information) we have put extra 2 assertions. First we needed to make sure that upon creation of the new schedule the API does create a corresponding schedule for it automatically. Second, we needed to make sure that newly added user is indeed in the group.

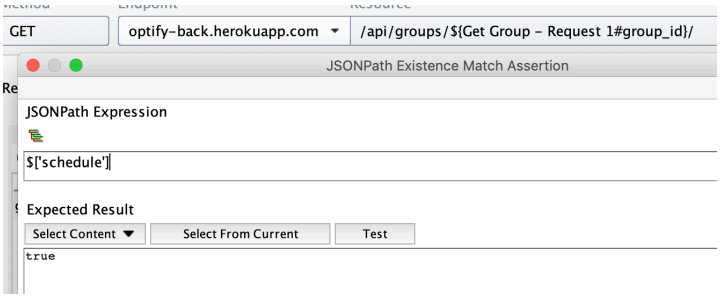


Figure 44 | Group Test Case Assertion 1

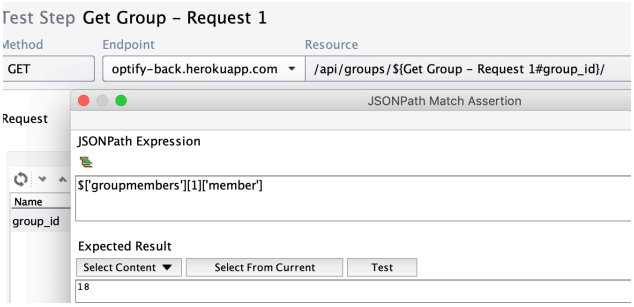


Figure 45 | Group Test Case assertion 2

Similarly at step 7 we needed to make sure that the user does not exist in the group, so one more existence assertion for that.

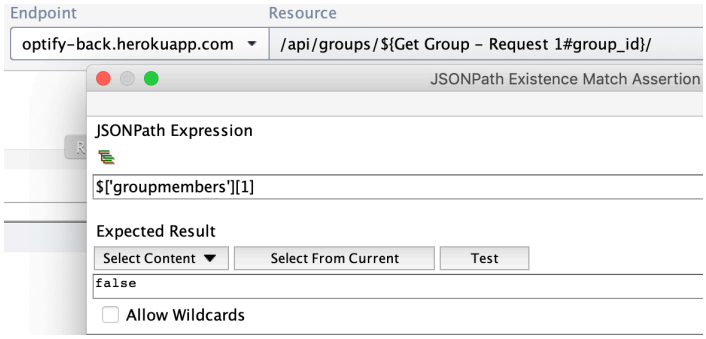


Figure 46 | Group Test Case Assertion 3

Upon running we can see that there are no errors.

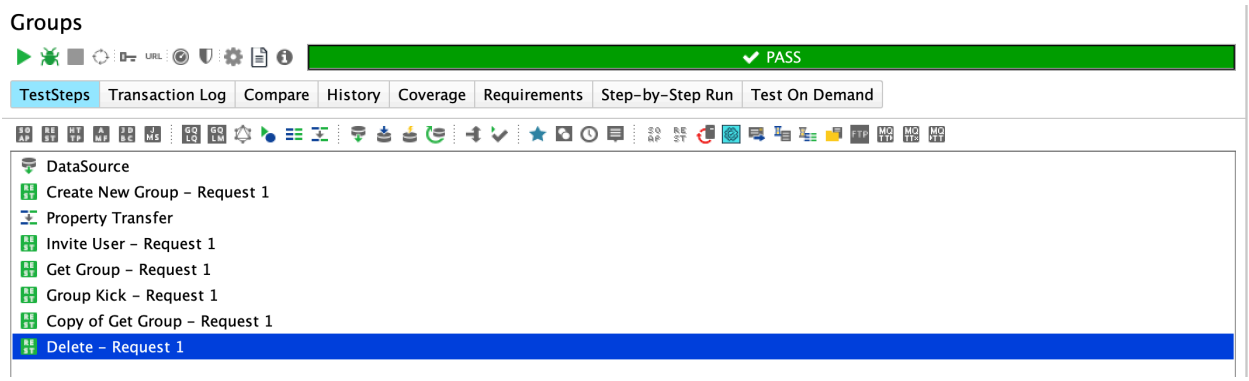


Figure 47 | Group Test Case Run

Front-End Application

For the unit testing of our Front-End application we were using the Android Studio and built in Dart language libraries. First, we had to configure the necessary dependencies for testing purposes. Afterward, test package was created, and test files were scripted.

1. Schedule Screen Test Case

•Participants**:** Orkhan, Rasul

•Pass/Fail Criteria: Schedule screen with Appbar and Month Schedule is created.

•Result: Pass

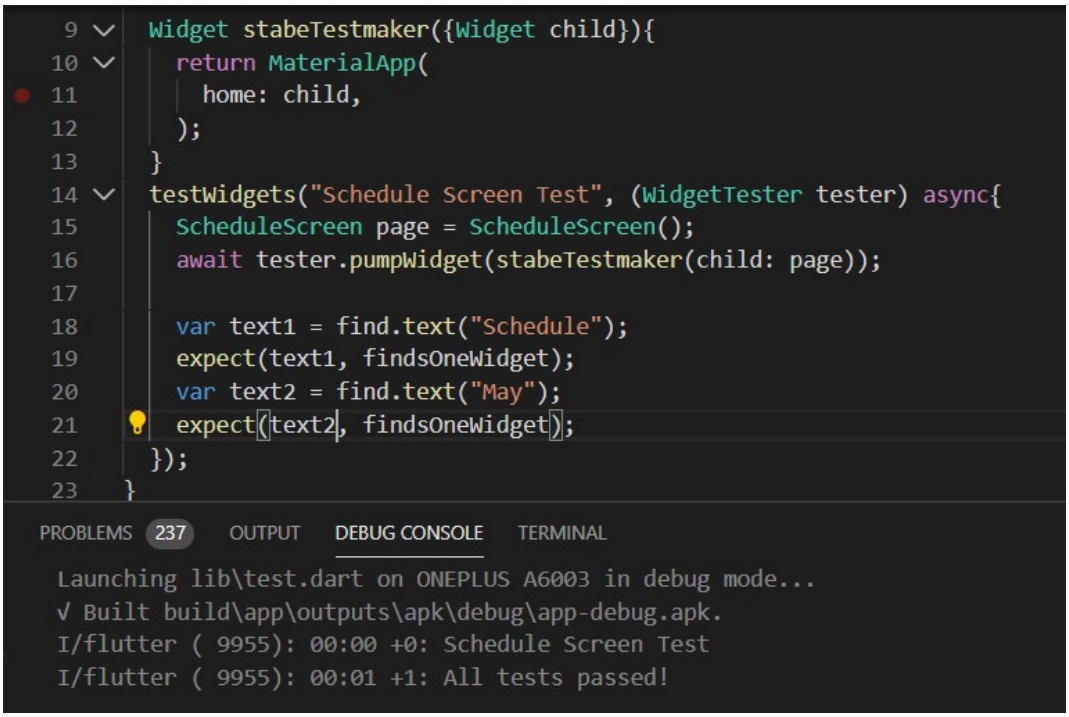


Figure 48|Schedule Screen test

2. Profile Screen Test Case

•Participants**:** Orkhan, Rasul

•Pass/Fail Criteria: Profile Screen with all necessary widgets is created, Appbar is created, the name and surname of the user are fetched from the backend and afterwards they match the real credentials of the user.

•Result: Fail then Pass

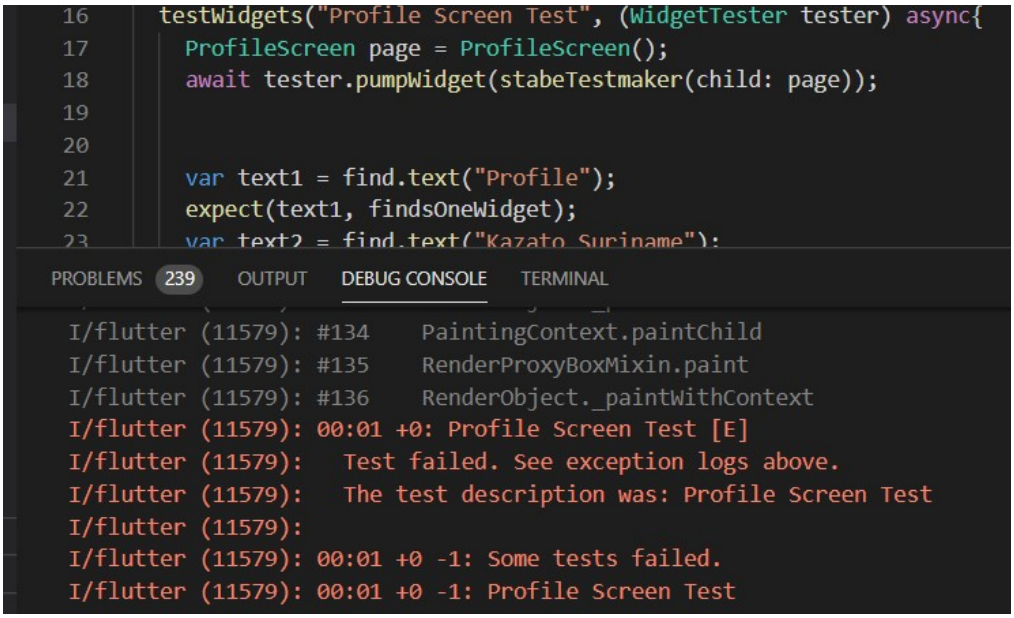


Figure 49|Profile Screen initial test

As you can see above, out first attempt of testing Profile Screen failed. We started investigating why it happened and later we figured out that it happened because ImageUrl of our Profile Screen was set incorrectly. Afterwards we proceeded to fixing it and running the enhanced test again. Test was passed.

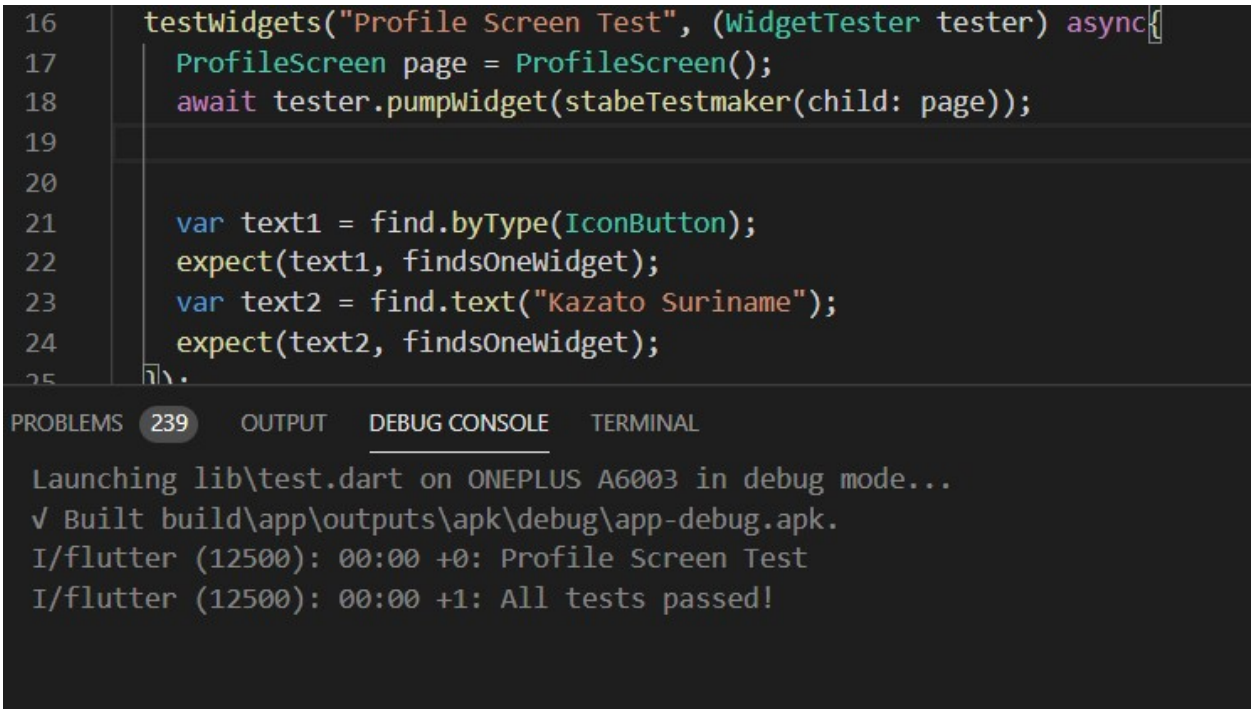


Figure 50|Profile screen test after fix

3. Chat Window Widget Test Case

•Participants: Orkhan, Rasul

•Pass/Fail Criteria: Chat Window created, and all message items added to ListView, TextField is created and camera icon is also drawn. Name of opposite person also written in AppBar.

•Result: Pass

```
17  testWidgets("Chat Window Test", (WidgetTester tester) async{
18    ChatWindow page = ChatWindow();
19    await tester.pumpWidget(stabeTestmaker(child: page));
20
21    var textField = find.byType(TextField);
22    expect(textField, findsOneWidget);
23    var iconButton = find.byIcon(Icons.camera_alt);
24    expect(iconButton, findsOneWidget);
25    var text2 = find.text("Orkhan Salahov");
26    expect(text2, findsOneWidget);
```

```
PROBLEMS 244   OUTPUT   DEBUG CONSOLE   TERMINAL
Launching lib\test.dart on ONEPLUS A6003 in debug mode...
√ Built build\app\outputs\apk\debug\app-debug.apk.
I/flutter (13888): 00:00 +0: Chat Window Test
I/flutter (13888): 00:01 +1: All tests passed!
```

Figure 51|Chat Window test

4. Chat Screen Test Case

•Participants**:** Orkhan, Rasul

•Pass/Fail Criteria: Chat Screen created, and all chat rooms added to ListView. Name of user and picture are also added to ListView.

•Result: Pass

```
18  testWidgets("Chat Screen Test", (WidgetTester tester) async{
19    ChatsScreen page = ChatsScreen();
20    await tester.pumpWidget(stabeTestmaker(child: page));
21
22    var text1 = find.text("Rasul Aliyev");
23    expect(text1, findsOneWidget);
24    var image = find.byType(CircleAvatar);
25    expect(image, findsOneWidget);
26    var text2 = find.text("Chats");
27    expect(text2, findsOneWidget);
28  });
```

```
PROBLEMS 255  OUTPUT  DEBUG CONSOLE  TERMINAL
Launching lib\test.dart on ONEPLUS A6003 in debug mode...
√ Built build\app\outputs\apk\debug\app-debug.apk.
D/DecorView(16659): onWindowFocusChangedFromViewRoot hasFocus: true, DecorView@c6516a6[MainActivity]
I/flutter (16659): 00:00 +0: Chat Screen Test
I/flutter (16659): 00:01 +1: All tests passed!
```

Figure 52|Chat Screen test

5. Groups Screen Test Case

•Participants**:** Orkhan, Rasul

•Pass/Fail Criteria: All group users of the current user are added to a ListView. My groups text is also added to the Appbar.

•Result: Pass

```
20    testWidgets("Groups Screen Test", (WidgetTester tester) async{
21      MembersUsers page = MembersUsers();
22      await tester.pumpWidget(stabeTestmaker(child: page));
23
24      var iconButton = find.byIcon(Icons.group);
25      expect(iconButton, findsOneWidget);
26      var text2 = find.text("My Groups");
27      expect(text2, findsOneWidget);
28    });
29  }
```

```
PROBLEMS 257  OUTPUT  DEBUG CONSOLE  TERMINAL
Launching lib\test.dart on ONEPLUS A6003 in debug mode...
√ Built build\app\outputs\apk\debug\app-debug.apk.
I/flutter (19349): 00:00 +0: Groups Screen Test
I/flutter (19349): 00:01 +1: All tests passed!
```

Figure 53|Group Screen Test Case

6. New Activity Screen Test Case

•Participants**:** Orkhan, Rasul

•Pass/Fail Criteria: All buttons, textfields added successfully.

•Result: Pass

```
18    testWidgets("New Activity Screen Test", (WidgetTester tester) async{
19      newActivityPersonal page = newActivityPersonal();
20      await tester.pumpWidget(stabeTestmaker(child: page));
21
22      var textformfield = find.byType(TextFormField);
23      expect(textformfield, findsOneWidget);
24
25      var text2 = find.text("Duration");
26      expect(text2, findsOneWidget);
27    });
28  }
```

```
PROBLEMS 255   OUTPUT   DEBUG CONSOLE   TERMINAL
Launching lib\test.dart on ONEPLUS A6003 in debug mode...
√ Built build\app\outputs\apk\debug\app-debug.apk.
I/flutter (18109): 00:00 +0: New Activity Screen Test
I/flutter (18109): 00:01 +1: All tests passed!
```

7. Contacts Screen Test Case

•Participants**:** Orkhan, Rasul

•Pass/Fail Criteria: All contacts of the current user are added to a ListView. Contacts text is also added to the Appbar.

•Result: Pass

```
19  testWidgets("Contacts Screen Test", (WidgetTester tester) async{
20    MemberPage page = MemberPage();
21    await tester.pumpWidget(stabeTestmaker(child: page));
22
23    var iconButton = find.byIcon(Icons.person);
24    expect(iconButton, findsOneWidget);
25    var text2 = find.text("Contacts");
26    expect(text2, findsOneWidget);
```

PROBLEMS 256 OUTPUT DEBUG CONSOLE TERMINAL

```
Launching lib\test.dart on ONEPLUS A6003 in debug mode...
√ Built build\app\outputs\apk\debug\app-debug.apk.
I/flutter (18949): 00:00 +0: Contacts Screen Test
I/flutter (18949): 00:01 +1: All tests passed!
```

Figure 54|Contacts Screen Test Case

2. System & Integration Testing

Since we are still in the phase of fully integrating the system our current integration testing consists of:

- Manually testing the features of the app in the debugger
- Running unit tests as a scenario.

Running Unit tests as scenario would basically mean to run sequentially all the unit tests described before. This would test the integration amongst those functionalities of the backend that we have unit tested. SoapUI provides a simple button to automate this process, results are as follows.

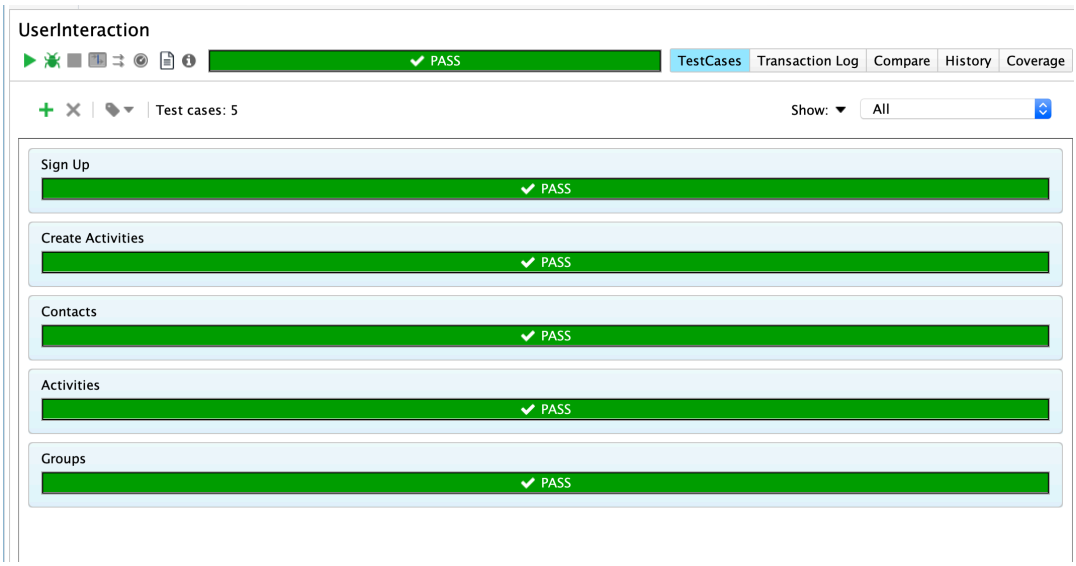


Figure 55 | Simple Integration testing

Step 1: Authenticate

Sign Up and Log In features of the application were tested. There were no problems occurring related to these screens. After authenticating the app must be refreshed automatically and the state of the is changed as expected.

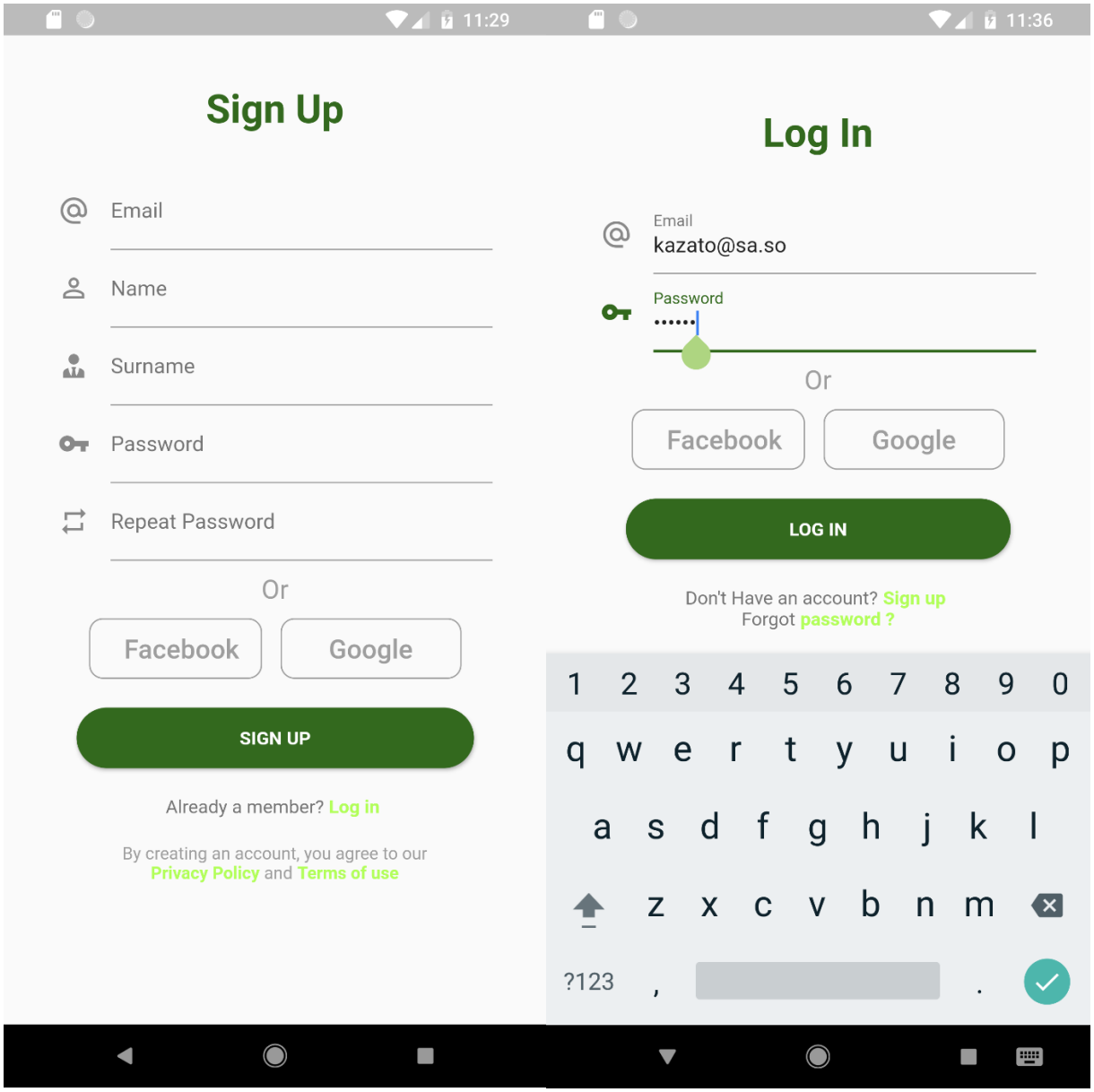


Figure 56|Authentication Screens

Step 2: Fetch Schedule Data and Display

After the sign in of the user to the application, the tabular form of the schedule is displayed to the user. All of the represented data correctly matches the expected data from the back end.

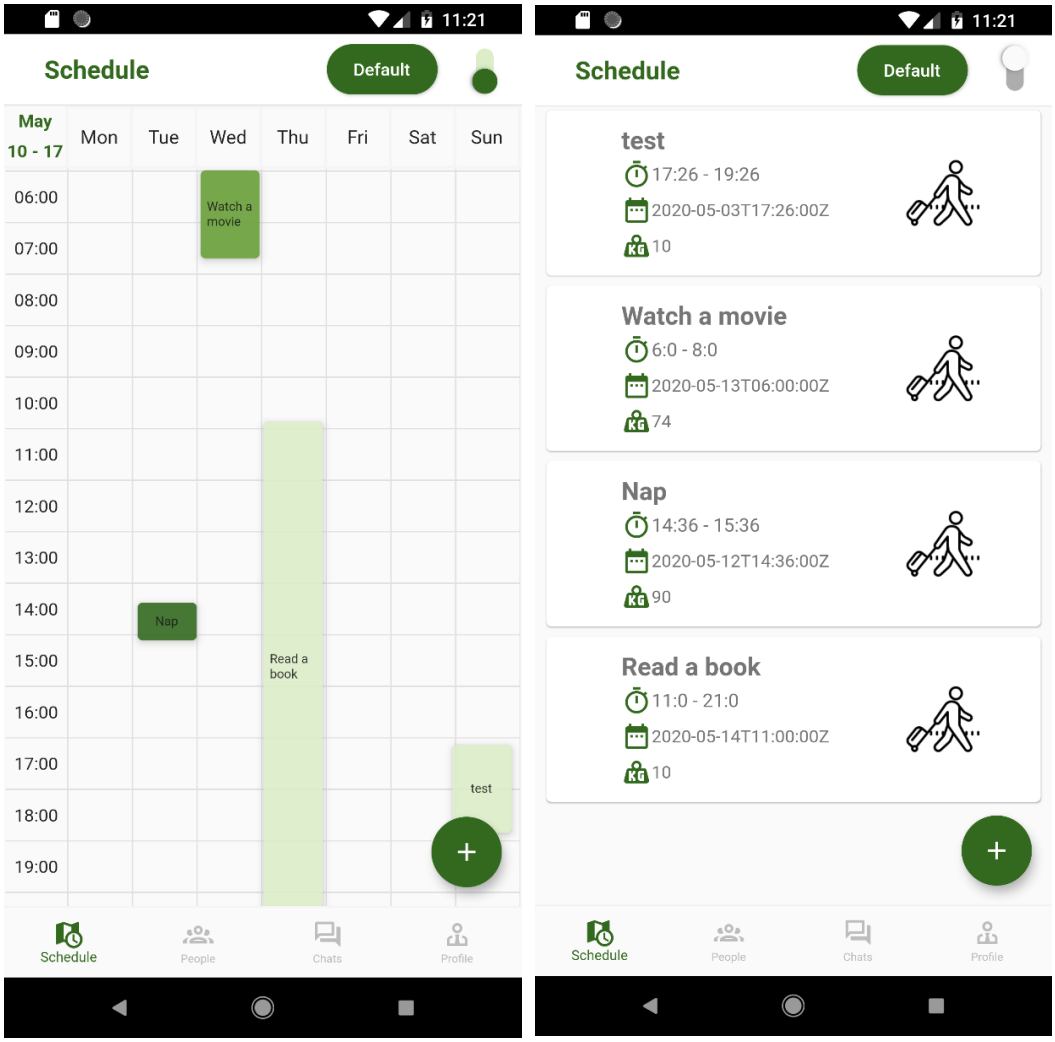


Figure 57|Schedule of the user initially

Step 3: Fetch User Data and Display

With the use of Bottom Navigation Bar the app state is navigated to Profile Screen.

All the necessary information is fetched from the back end and stored in the memory of the device, thereafter it is displayed as expected.

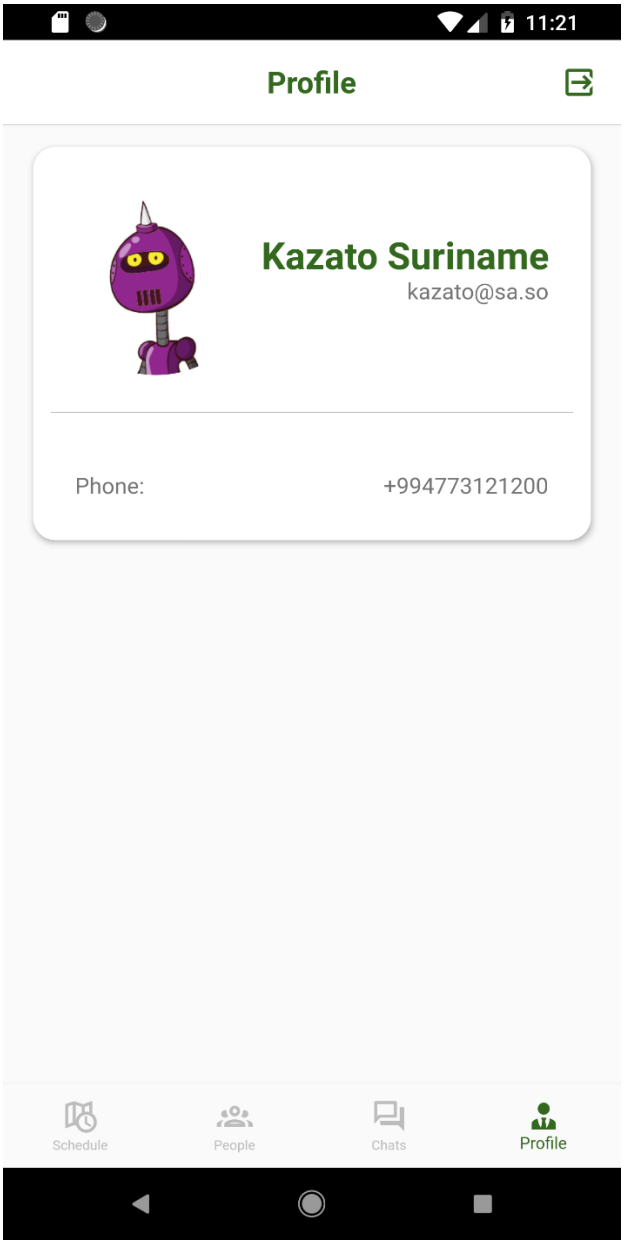


Figure 58|Profile of the User

Step 4: Add a New Activity and Post it to Back-End

From the Schedule Screen the app state is navigated to New Activity Addition screen with the use of floating action button. The application takes the input from the user as expected, thereafter the post request to backend is sent. When we check the back end we can observe the expected behavior, which means that the operation was successful. Thereafter application prompts the success message.

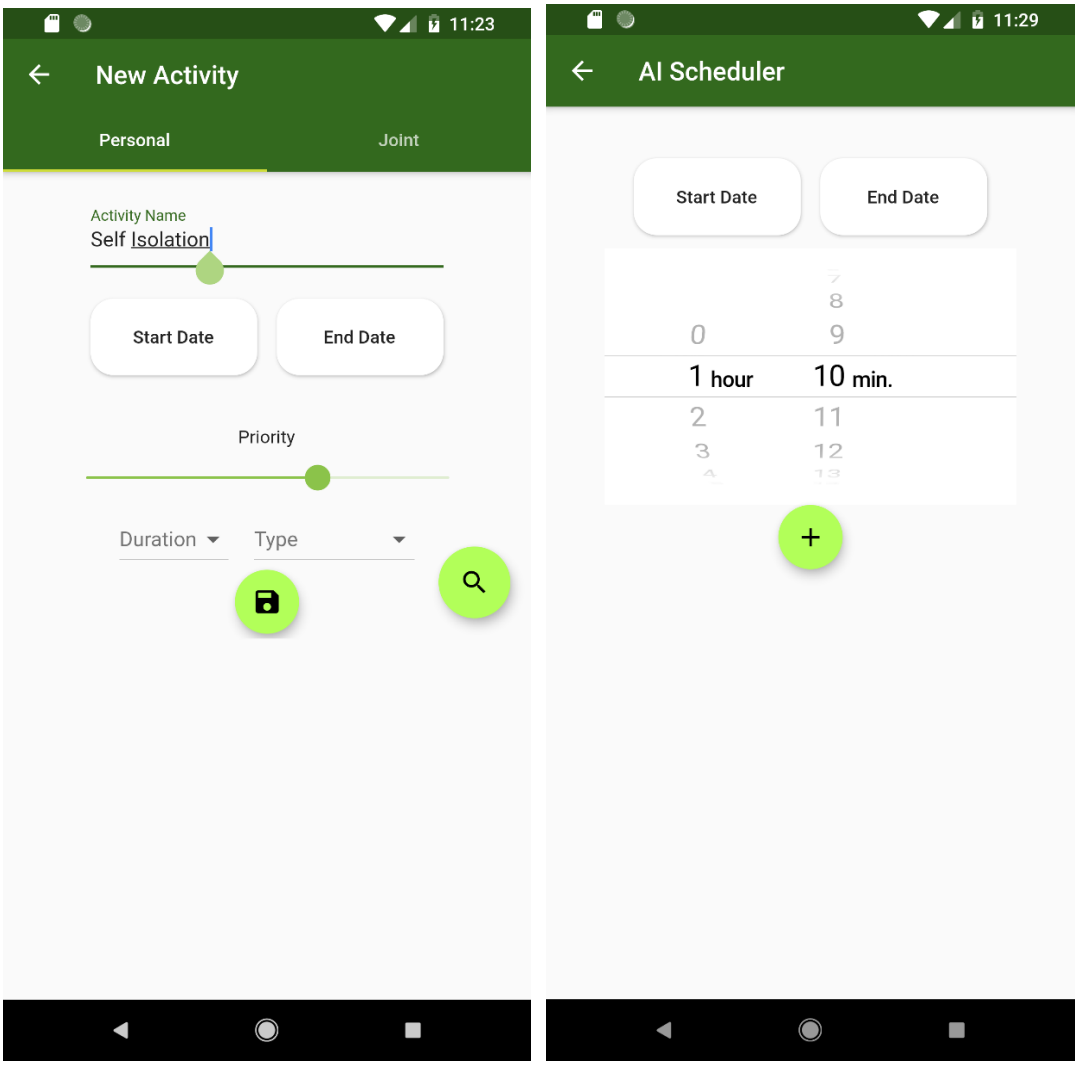


Figure 59|Addition of new Activity

Step 5: Display newly added Activity in Schedule Screen

After the addition of the new activity, Application is navigated to the schedule screen, where all the activities are displayed correctly, including the newly added one.

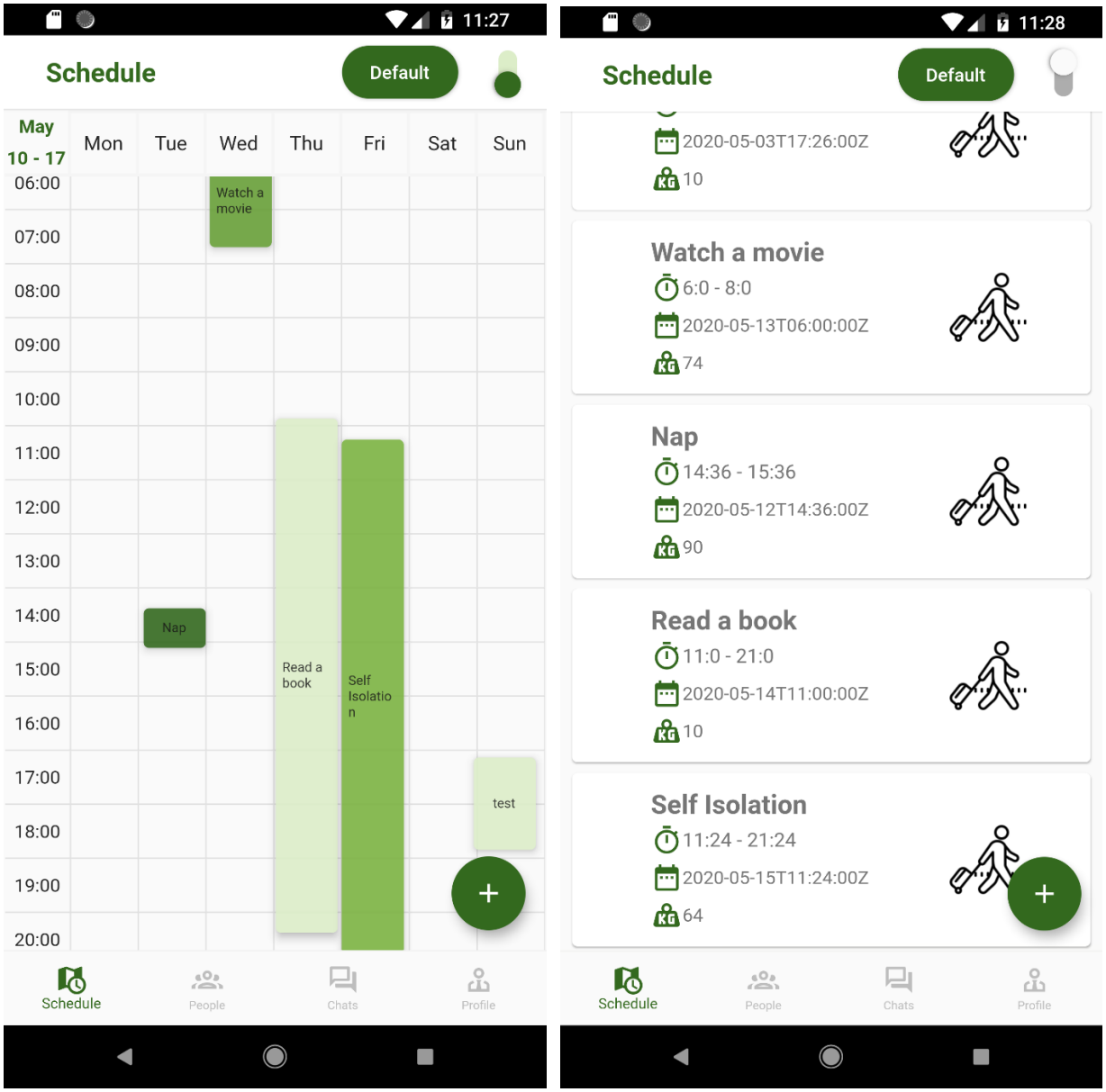


Figure 60|Schedule Screen after insertion of Activity

Step 6: Log out of user

The final step of our integration testing's scenario is logging out of the application. It is pretty easy to do and as expected, there are no problems occurring here. After the log out operation the application state is changed to initial log in screen.

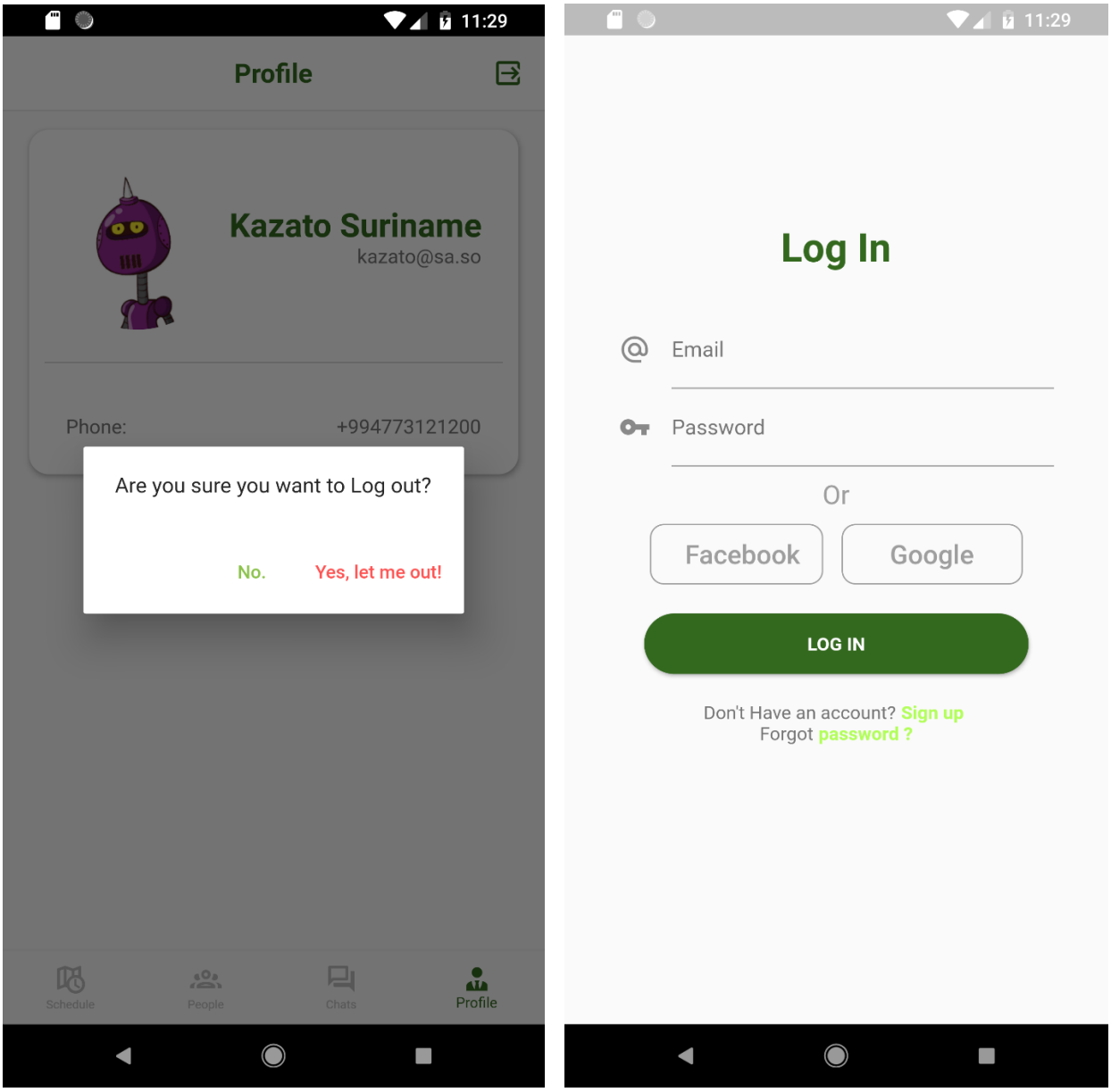


Figure 61|Logging out of the user

3. Performance & Stress Testing

For stress testing we have been reading the web for acceptable delays but decided to go with our intuition, since at the end of the day the operation performed does affect how much delay the user expects from the service. So, there is no solid line. We will continue performing stress tests on the Test Cases mentioned before for the Back-End. For all the stress tests though we are going to have **VU = 10** (Virtual Users)/sec. That is the number of simultaneous requests.

Sign Up

For sign up we chose to have the error threshold as 2000ms. So, any request exceeding that amount is going to be considered as failed.

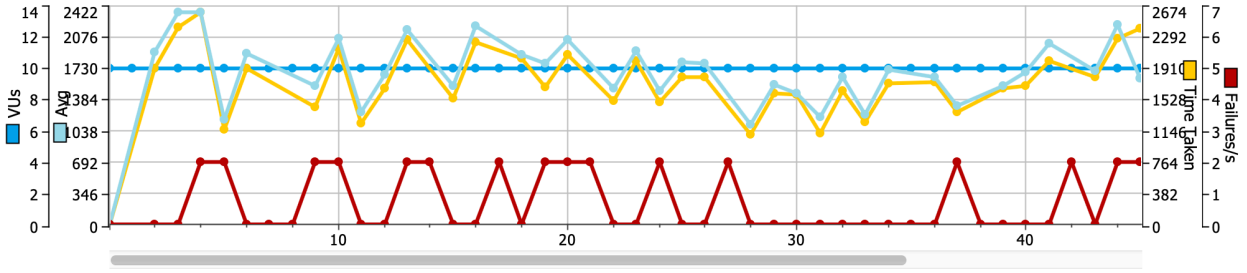


Figure 62 | Sign Up Performance Test

We can see that there were couple of requests every now and then.

Create Activities

For create activities we chose our threshold as 1500ms.

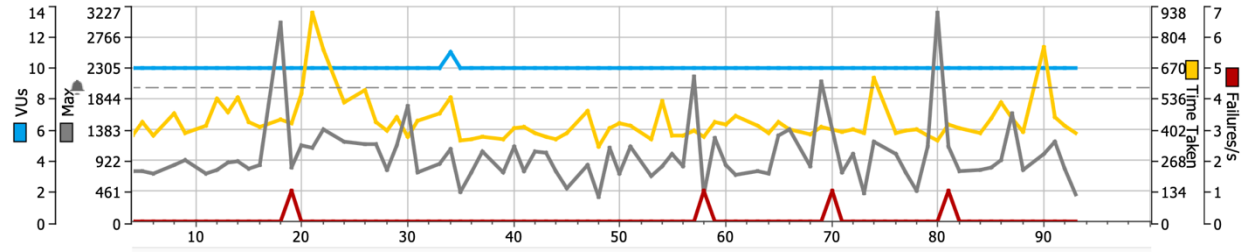


Figure 33 |Create Activities Performance Test

For Create activities the results are more promising.

Activities View

For viewing activities, we set the max threshold as 1second, since it is a small transaction.

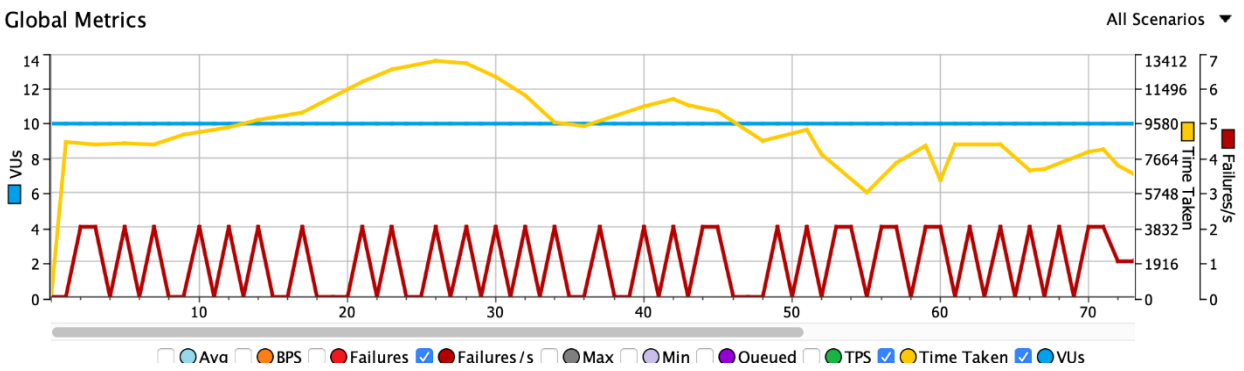


Figure 34 |View Activities Performance Test

This one was quite interesting; performance was unexpectedly bad. This one might need to be converted to raw SQL later, since Django's default serialization might be poor in performance for nested structures.

Group

Groups, being the most complicated transaction, has demanded quite high latency, therefore we've set it to 3 seconds.

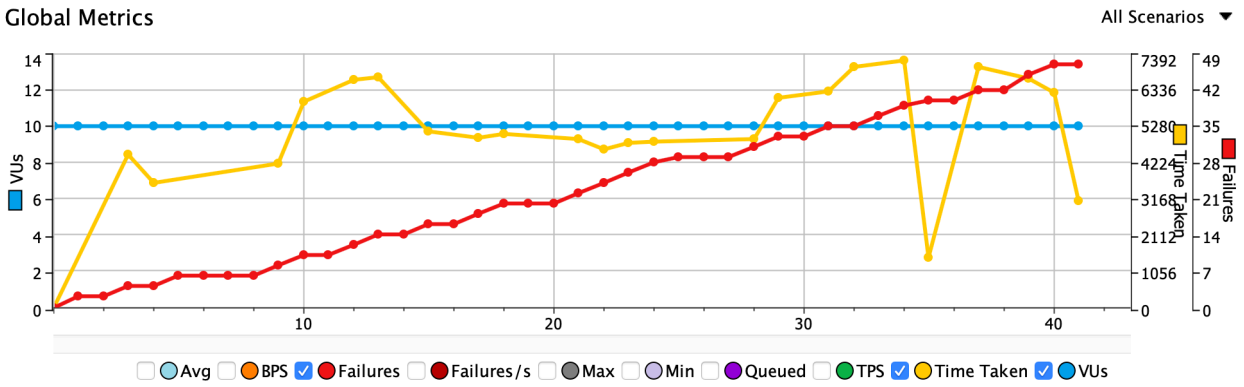


Figure 35 |Group Performance Test

4. Security Testing

For security Testing we didn't expect much vulnerability to start with, since we were using framework that has been enterprise tested. We have relied on SoapUI to automate all kinds of security tests for us. Those tests are

- Boundary Scan
- Cross Site Forgery
- Fuzzy Scan
- Invalid Types
- SQL Injection
- XPath Injection
- HTTP Method Fuzzing
- Sensitive Files Exposure
- Weak Authentication

The results were quite expected, as no vulnerabilities were found, except for HTTP Method Fuzzing.

Total scans performed: 258

Issues found: 3

Scan	Issues Found In Test Steps		Total Issues Found
HTTP Method Fuzzing	Schedule View	3	3

Figure 63 | Security Test Result Summary

This method was OPTION method that was being automatically defined by one serialization framework that we were using. It is was quite a novelty for us, but this was merely a warning not an error.

HTTP Method Fuzzing

An HTTP Method Fuzzing Scan attempts to use other HTTP verbs (methods) than those defined in an API. For instance, if you have defined GET and POST, it will send requests using the DELETE and PUT verbs, expecting an appropriate HTTP error response and reporting alerts if it doesn't receive it.

Sometimes, unexpected HTTP verbs can overwrite data on a server or get data that shouldn't be revealed to clients.

Scan	HTTP Method Fuzzing
Severity	WARNING
Endpoint	https://optify-back.herokuapp.com/api/schedules/30/activities/
Request	OPTIONS https://optify-back.herokuapp.com/api/schedules/30/activities/ HTTP/1.1
Test Step	Schedule View
Modified Parameters	**Name** / **Value**: method / OPTIONS
Response	**Content-type:** application/json **Content length:** 1928 **Response is too big. Beginning of the response:**

Figure 64 | Security Test Output

7. Conclusion & References

7.1 Conclusion

As a result of this project the team members gained multiplicity of different skills which are believed to be highly beneficial for any kind developers. Those skills are both software oriented and interpersonal. Aside from gaining those skills, we , as a group, managed to come up with a software about which we are positive that will be used by people. With the help of the application and the algorithms it uses, users will benefit from it and optimize their schedule to balance all parts of their life.

7.2 Conclusion

1. *http://django-rest-framework.org*
2. *http://flutter.dev*
3. *http://github.com*
4. *http://slack.com*
5. *https://calendar.google.com/*
6. *https://dart.dev*
7. *https://www.iana.org/time-zones*
8. *https://www.timeanddate.com/calendar/days/monday.html*
9. *https://www.workast.com*
10. *https://github.com/Tendrl/documentation/wiki/Best-Practices-for-Response-Times-and-Latency*
11. *https://github.com/joke2k/faker*
12. *https://www.soapui.org*
13. *https://flutter.dev*